Patrick Rodler

Interactive Debugging of Knowledge Bases

DISSERTATION

submitted in fulfilment of the requirements for the degree of

Doktor der Technischen Wissenschaften

Alpen-Adria-Universität Klagenfurt

Fakultät für Technische Wissenschaften

Mentor
O.Univ.-Prof. Dipl.-Ing. Dr. Gerhard Friedrich
Alpen-Adria-Universität Klagenfurt
Institut für Angewandte Informatik

1st Evaluator
O.Univ.-Prof. Dipl.-Ing. Dr. Gerhard Friedrich
Alpen-Adria-Universität Klagenfurt
Institut für Angewandte Informatik

2nd Evaluator
Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Wotawa
TU Graz
Institut für Softwaretechnologie

Klagenfurt, November 2015

Affidavit

I hereby declare in lieu of an oath that

- the submitted academic paper is entirely my own work and that no auxiliary materials have been used other than those indicated,
- I have fully disclosed all assistance received from third parties during the process of writing the paper, including any significant advice from supervisors,
- any contents taken from the works of third parties or my own works that have been included either literally or in spirit have been appropriately marked and the respective source of the information has been clearly identified with precise bibliographical references (e.g. in footnotes),
- to date, I have not submitted this paper to an examining authority either in Austria or abroad and that
- the digital version of the paper submitted for the purpose of plagiarism assessment is fully consistent with the printed version.

I am aware that a declaration contrary to the facts will have legal consequences.

(Signature)   (Place, date)

Contents

List of Figures

List of Tables

Abstract

Most artificial intelligence applications rely on knowledge about a relevant real-world domain that is encoded in a knowledge base (KB) by means of some logical knowledge representation language. The most essential benefit of such logical KBs is the opportunity to perform automatic reasoning to derive implicit knowledge or to answer complex queries about the modeled domain. The feasibility of meaningful reasoning requires a KB to meet some minimal quality criteria such as consistency; that is, there must not be any contradictions in the KB. Without adequate tool assistance, the task of resolving such violated quality criteria in a KB can be extremely hard even for domain experts, especially when the problematic KB includes a large number of logical formulas, comprises complicated formalisms, was developed by multiple people or in a distributed fashion or was (partially) generated by means of some automatic systems.

Non-interactive debugging systems published in research literature often cannot localize all possible faults (*incompleteness*), suggest the deletion or modification of unnecessarily large parts of the KB (*non-minimality*), return incorrect solutions which lead to a repaired KB not satisfying the imposed quality requirements (*unsoundness*) or suffer from *poor scalability* due to the inherent complexity of the KB debugging problem. Even if a system is complete and sound and considers only minimal solutions, there are generally exponentially many solution candidates to select one from. However, any two repaired KBs obtained from these candidates differ in their semantics in terms of entailments and non-entailments. Selection of just any of these repaired KBs might result in unexpected entailments, the loss of desired entailments or unwanted changes to the KB which in turn might cause unexpected new faults during the further development or application of the repaired KB. Also, manual inspection of a large set of solution candidates can be time-consuming (if not practically infeasible), tedious and error-prone since human beings are normally not capable of fully realizing the semantic consequences of deleting a set of formulas from a KB. Hence there is a need for adequate tools that support a user when facing a faulty KB.

In this work, we account for these issues and propose methods for the interactive debugging of KBs which are complete and sound and compute only minimally invasive solutions, i.e. suggest the deletion or modification of just a set-minimal subset of the formulas in the problematic KB. User interaction takes place in the form of queries asked to a person, e.g. a domain expert, about intended and non-intended entailments of the correct KB. To construct a query, only a minimal set of two solution candidates must be available. After the answer to a query is known, the search space for solutions is pruned. Iteration of this process until there is only a single solution candidate left yields a repaired KB which features exactly the semantics desired and expected by the user.

The novel contributions of this work are:

- *Thorough Theoretical Workup of the Topic of Interactive Debugging of Monotonic KBs:* We evolve the theory of the topic by first elaborating on the theory of non-interactive KB debugging, revealing crucial shortcomings in the application of non-interactive methods and thereby motivating the development and deployment of interactive approaches in KB debugging. Then, we give some important results that guarantee the feasibility of interactive KB debugging, give some precise definitions of the problems interactive KB debugging aims to solve and present algorithms that provably solve these problems.

- *A Complete Picture of an Interactive Debugging System is Drawn:* This is the first work that deals with an entire system of algorithms that are required for the interactive debugging of monotonic KBs, considers and details all algorithms separately, proves their correctness and demonstrates how all these algorithms are orchestrated to make up a full-fledged and provably correct interactive KB debugging system.

- *Two New Algorithms for the Iterative Computation of Candidate Solutions* in the scope of interactive KB debugging are proposed. The first one guarantees constant convergence towards the exact solution of the interactive KB problem by the ascertained reduction of the number of remaining solutions after any query is answered. The second one features powerful search tree pruning techniques and might thus be expected to exhibit a more time- and space-saving behavior than existing algorithms, in particular for growing problem instances.

- *Suggestion and Extensive Analysis of Different Methods for Selection of the "Best" Query* to ask the user next. We compare a greedy "split-in-half" strategy that proposes queries which eliminate half of the known candidate solutions with a strategy relying on information entropy that chooses the query with highest information gain based on (a user's) beliefs about faults in the KB. Comprehensive experiments manifest that an average guess of the fault information suffices to reduce the query answering effort for the interacting user, often to a significant extent, by means of the latter strategy compared to the former. Moreover, we demonstrate that both methods clearly outperform a random way of selecting queries.

- *Presentation of a Reinforcement Learning Query Selection Strategy.* Minimal effort for the interacting user can be achieved if both the query selection method is chosen carefully and the provided fault information satisfies some minimum quality requirements. In particular, for deficient fault information and unfavorable strategy for query selection, we observe cases where the overhead in terms of user effort exceeds 2000% (!) in comparison to employing a more favorable query selection strategy. Since, unfortunately, assessment of the fault information is only possible a-posteriori (after the debugging session is finished and the correct solution is known), we devise a learning strategy (RIO) that continuously adapts its behavior depending on the performance achieved and in this vein minimizes the risk of using low-quality fault information. This approach makes interactive debugging practical even in scenarios where reliable fault estimates are difficult to obtain. Evaluations provide evidence that for 100% of the cases in the hardest (from the debugging point of view) class of faulty test KBs, RIO performed at least as good as the *best* other strategy and in more than 70% of these cases it even manifested superior behavior to the *best* other strategy. Choosing RIO over other approaches can involve an improvement by the factor of up to 23, meaning that more than 95% of user time and effort might be saved per debugging session.

- *Provisioning of Mechanisms for Efficiently Dealing with KB Debugging Problems Involving High Cardinality Faults.* In the standard interactive debugging approach described in this work, the computation of queries is based on the generation of the set of *most probable* solution candidates. By this postulation, certain quality guarantees about the output solution can be given. However, we learn that dropping this requirement can bring about substantial savings in terms of time and especially space complexity of interactive debugging, in particular in debugging scenarios where faulty KBs are (partly) generated as a result of the application of automatic systems. In such situations, we propose to base query computation on *any* set of solution candidates using a "direct" method for candidate generation. We study the application of this direct method to high cardinality faults in KBs and find out that the number of required queries per debugging session is scarcely affected for cases when the standard approach is also applicable. However, the direct method proves applicable in situations when the standard approach is not (due to time or memory issues) and is still able to locate the correct solution.

Part I

Prolog

In this part, we first give an introduction in Chapter 1. This includes a motivation why knowledge base debugging is a "hot topic" (and even getting hotter as intelligent applications and devices become more and more ubiquitous), an introduction to the non-interactive debugging of knowledge bases and the revealment of decisive shortcomings of this paradigm, e.g. poor scalability and the risk of obtaining solutions of inferior quality. As a solution to the identified issues we then explain how a (group of) user(s) might collaborate with an interactive debugging system to determine high-quality solutions even in scenarios where non-interactive systems fail. Further, we discuss the design and the components of a generic interactive debugger, provide an illustrating example and outline the powerful feature of our system to be able to incorporate background knowledge into the debugging process which can drastically reduce the search space for solutions and disclose faults in the knowledge base that could be missed otherwise. Finally, we provide an enumeration of the contributions of this work and discuss the further organization of this part and of the rest of this work.[1]

---

[1]Parts of Part I already appeared in [Rod15].

Chapter 1

Introduction

Motivation. Most artificial intelligence applications rely on knowledge that is encoded in a knowledge base (KB) by means of some logical knowledge representation language such as propositional logic (PL) [CL73], Datalog [CGT89], first-order logic (FOL) [CL73], The Web Ontology Language (OWL [PSHH$^+$04], OWL 2 [GHM$^+$08, MPSP09]) or Description Logic (DL) [BCM$^+$07]. Experts in a variety of application domains keep developing KBs of constantly growing size. A concrete example of a repository containing biomedical KBs is the Bioportal[2], which comprises vast ontologies with tens or even hundreds of thousands of terms each (e.g. the SNOMED-CT ontology with currently over 395.000 terms). Such KBs however pose a significant challenge for people as well as tools involved in their evolution, maintenance and application.

All these activities are based on the most essential benefit of logical KBs, namely the opportunity to perform automatic reasoning to derive implicit knowledge or to answer complex queries about the modeled domain. The feasibility of meaningful reasoning requires a KB to meet the minimum quality criterion *consistency*, i.e. there must not be any contradictions in the KB. Because *any* logical formula can be derived from an inconsistent KB. Further on, one might postulate further requirements to be met by a KB. For instance, one might consider faulty a FOL KB entailing $\forall X \, \neg p(X)$ for some predicate symbol p occurring in the KB. Such a KB would be incoherent, i.e. it would violate the requirement *coherency* (which was originally defined for DL KBs [SHCH07, PSK05]). Additionally, test cases can be specified giving information about desired (*positive test cases*) and non-desired (*negative test cases*) entailments a correct KB should feature. This characterization of a KB's intended semantics is a direct analogon to the field of software debugging, where test cases are exploited as a means to verify the correct semantics of the program code.

As KBs are growing in size and complexity, their likeliness of violating one of these criteria increases. Faults in KBs may, for instance, arise because human reasoning is simply overstrained [HBP11, HPS09]. That is, generally a person will not be capable of completely grasping or mentally processing the entire knowledge contained in a (large or complex) KB at once. In fact, a person might fully comprehend some isolated part of a the KB, but might not be able to determine or understand all implications or non-implications of this isolated part combined with other parts of a KB, i.e. when new logical formulas are added.

Another reason for the non-compliance with the mentioned quality criteria imposed on KBs might be that multiple (independently working) editors contribute to the development of the KB [NCLM06] which may lead to contradictory formulas. The OBO Project[3] and the NCI Thesaurus[4] are examples of

---

[2]http://bioportal.bioontology.org
[3]http://obo.sourceforge.net
[4]http://nciterms.nci.nih.gov/ncitbrowser

collaborative KB development projects. Employing automatic tools, e.g. [JRG11, NB12, JMSK09], to generate (parts of) KBs can further exacerbate the task of KB quality assurance [Mei11, EFvH$^+$11].

Moreover, as studies in cognitive psychology [CP71, JL99] attest, humans make systematic errors while formulating or interpreting logical formulas. These observations are confirmed by [RDH$^+$04, RCVB09] which present common faults people make when developing a KB (ontology). Hence, it is essential to devise methods that can efficiently identify and correct faults in a KB.

Non-Interactive KB Debugging. Given a set of requirements to the KB and sets of test cases, KB debugging methods [SHCH07, KPHS07, FS05, HPS08] can localize a (potential) fault by computing a subset $\mathcal{D}$ of the formulas in the KB $\mathcal{K}$ called a *diagnosis*. At least all formulas in a diagnosis must be (adequately) modified or deleted in order to obtain a KB $\mathcal{K}^*$ that satisfies all postulated requirements and test cases. Such a KB $\mathcal{K}^*$ constitutes the solution to the *KB debugging problem*. Figure 1.1[5] outlines such a KB debugging system. The input to the system is a *diagnosis problem instance (DPI)* defined by

- some KB $\mathcal{K}$ formulated using some (monotonic) logical language $\mathcal{L}$ (every formula in $\mathcal{K}$ might be correct or faulty),

- (optionally) some KB $\mathcal{B}$ (over $\mathcal{L}$) formalizing some background knowledge relevant for the domain modeled by $\mathcal{K}$ (such that $\mathcal{B}$ and $\mathcal{K}$ do not share any formulas; all formulas in $\mathcal{B}$ are considered correct)

- a set of requirements R to the correct KB,

- sets of positive (P) and negative (N) test cases (over $\mathcal{L}$) asserting desired semantic properties of the correct KB and

- (optionally) some fault information **FP**, e.g. in terms of fault probabilities of logical formulas in $\mathcal{K}$.

Moreover, the system requires a sound and complete logical reasoner for deciding consistency (coherency) and calculating logical entailments of a KB formulated over the language $\mathcal{L}$. Some approaches (including the ones presented in this work) use the reasoner as a black-box (e.g. [SFFR12, Hor11]) within the debugging system. That is, the reasoner is called as is and serves as an oracle independent from other computations during the debugging process; that is, the internals of the reasoner are irrelevant for the debugging task. On the other hand, glass-box approaches (e.g. [SHCH07, Hor11, KPSH05]) attempt to exploit internal modifications of the reasoner for debugging purposes; in other words, the sources of problems (e.g. contradictory formulas) in the KB are computed as a direct consequence of reasoning [Hor11]. The advantages of a black-box approach over a glass-box approach are the lower memory consumption and better performance [KPSH05] of the reasoner and the reasoner independence of the debugging method. The latter benefit is essential for the generality of our approaches and their applicability to various knowledge representation formalisms.

Given these inputs, the debugging system focuses on (a subset of) all possible fault candidates (usually the set of minimal, i.e. irreducible, diagnoses) and usually outputs the most probable one amongst these if some fault information is provided or the minimum cardinality one, otherwise. Alternatively, a debugging system might also be employed to calculate a predefined number of (most probable or minimum cardinality) minimal diagnoses or to determine all minimal diagnoses computable within a predefined time limit.

---

[5]Thanks to Kostyantyn Shchekotykhin for making available to me parts of this diagram.
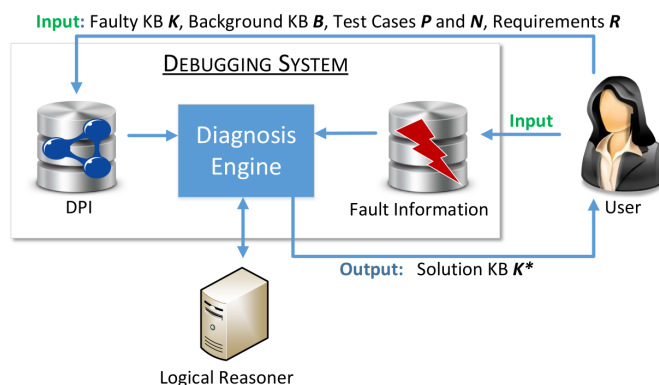
Figure 1.1: The principle of non-interactive KB debugging.

Issues with Non-Interactive KB Debugging Systems. In real-world scenarios, debugging tools often have to cope with large numbers of minimal diagnoses where the trivial application, i.e. deletion, of any minimal diagnosis leads to a (repaired) KB with different semantics in terms of entailed and non-entailed formulas. For example, in [SF10] a sample study of real-world KBs revealed that the number of different minimal diagnoses might exceed thousand by far (1782 minimal diagnoses for a KB with only 1300 formulas). In such situations simple visualization of all these alternative modifications of the ontology is clearly ineffective. Selecting a wrong diagnosis (in terms of its semantics, *not* in terms of fulfillment of test cases and requirements) can lead to unexpected entailments or non-entailments, lost desired entailments and surprising future faults when the KB is further developed. Manual inspection of a large set of (minimal) diagnoses is time-consuming (if not practically infeasible), error-prone and often computationally infeasible due to the complexity of diagnosis computation.

Moreover, [Stu08] has put several (non-interactive) debugging systems to the test using a test set of faulty (incoherent OWL) real-world KBs which were partly designed by humans and partly by the application of automatic systems. The result was that most of the investigated systems had serious performance problems, ran out of memory, were not able to locate all the existing faults in the KB (incompleteness), reported parts of a KB as faulty which actually were not faulty (unsoundness), produced only trivial solutions or suggested non-minimal faults (non-minimality). Often, performance problems and incompleteness of non-interactive debugging methods can be traced back to an explosion of the search tree for minimal diagnoses.

The Solution: Interactive KB Debugging. In this work we present algorithms for interactive KB debugging. These aim at the gradual reduction of compliant minimal diagnoses by means of user interaction, thereby seeking to prevent the search tree for minimal diagnoses from exploding in size by performing regular pruning operations. "User" in this case might refer to a single person or multiple persons, usually experts of the particular domain the faulty KB is dealing with such as biology, medicine or chemistry. Throughout an interactive debugging session, the user is asked a set of automatically chosen queries about the domain that should be modeled by a given faulty KB. A query can be created by the system after a set $\mathbf{D}$ of a minimum of two minimal diagnoses has been precomputed (we call $\mathbf{D}$ the *leading diagnoses*). Each query is a conjunction (i.e. a set) of logical formulas that are entailed by some correct subset of the formulas in the KB. With regard to one particular query Q, any set of minimal diagnoses for the KB, in particular the set $\mathbf{D}$ which has been utilized to generate Q, can be partitioned into three sets, the first one ($\mathbf{D}^+$) including all diagnoses in $\mathbf{D}$ compliant only with a positive answer to Q, the second ($\mathbf{D}^-$) including all diagnoses in $\mathbf{D}$ compliant only with a negative answer to Q, and the third ($\mathbf{D}^0$) including all

diagnoses in **D** compliant with both answers. A positive answer to Q signalizes that the conjunction of formulas in Q must be entailed by the correct KB wherefore Q is added to the set of positive test cases. Likewise, if the user negates Q, this is an indication that at least one formula in Q must not be entailed by the correct KB. As a consequence, Q is added to the set of negative test cases.

Assignment of a query Q to either set of test cases results in a new debugging scenario. In this new scenario, all elements of $\mathbf{D}^-$ are no longer minimal diagnoses given that Q has been classified as a positive test case. Otherwise, all diagnoses in $\mathbf{D}^+$ are invalidated. In this vein, the successive reply to queries generated by the system will lead the user to the single minimal solution diagnosis that perfectly reflects their intended semantics. In other words, after deletion of all formulas in the solution diagnosis from the KB and the addition of the conjunction of all formulas in the specified positive test cases to the KB, the resulting KB meets all requirements and positive as well as negative test cases. In that, the added formulas contained in the positive test cases serve to replace the desired entailments that are broken due to the deletion of the solution diagnosis from the KB.

Thence, in the interactive KB debugging scenario the user is not required to cope with the understanding of which faults (e.g. sources of inconsistency or implications of negative test cases) occur in the faulty initial KB, why they are faults (i.e. why particular entailments are given and others not) and how to repair them. All these tasks are undertaken by the interactive debugging system.

The proposed approaches to interactive KB debugging in this work follow the standard *model-based diagnosis (MBD)* technique [Rei87, dKW87]. MBD has been successfully applied to a great variety of problems in various fields such as robotics [SW05], planning [SW09], debugging of software programs [WSM02], configuration problems [FFJS04], hardware designs [FSW99], constraint satisfaction problems and spreadsheets [ARW12]. Given a description (model) of a system, together with an observation of the system's behavior which conflicts with the intended behavior of the system, the task of MBD is to find those components of the system (a diagnosis) which, when assumed to be functioning abnormally, provide an explanation of the discrepancy between the intended and the observed system behavior. Translated to the setting of KB debugging, the set of "system components" comprises the formulas ax_i in the given faulty KB $\mathcal{K}$. The "system description" refers to the statement that the KB $\mathcal{K}$ along with the background KB $\mathcal{B}$ and the positive test cases $p \in P$ must meet all predefined requirements (e.g. consistency, coherency) and must not logically entail any of the negative test cases $n \in N$, i.e.

(i) $\mathcal{K} \cup \mathcal{B} \cup \bigcup_{p \in P} p$ satisfies requirement r for all $r \in R$ and

(ii) $\mathcal{K} \cup \mathcal{B} \cup \bigcup_{p \in P} p \not\models n$ for all $n \in N$.

The "observation which conflicts with the intended behavior of the system" corresponds to the finding that (i) or (ii) or both are violated. That is, the "system description" along with the "observation" and the assumption that all components are sound yields an inconsistency. An "explanation for the discrepancy between observed and intended system behavior" (i.e. a diagnosis) is the assumption $\mathcal{D}$ that all formulas in a subset $\mathcal{D}$ of $\mathcal{K}$ are faulty ("behave abnormally") and all formulas in $\mathcal{K} \setminus \mathcal{D}$ are correct ("do not behave abnormally") such that the "system description" along with the "observation" and the assumption $\mathcal{D}$ is consistent. Computation of (minimal) diagnoses is accomplished with the aid of *minimal conflict sets*, i.e. irreducible sets of formulas in the KB $\mathcal{K}$ that preserve the violation of (i) or (ii) or both.

An MBD problem can be modeled as an abduction problem [BATJ91], i.e. finding an explanation for a set of data. It was proven in [BATJ91] that the computation of the first explanation (minimal diagnosis) is in P. However, given a set of explanations (minimal diagnoses) it is NP-complete to decide whether there is an additional explanation (minimal diagnosis). Stated differently, the detection of the first explanation can be efficiently accomplished whereas the finding of any further one is intractable (unless P = NP). When seeing the (interactive) KB debugging problem as an abduction problem, one must additionally take into account the costs for reasoning. Because, a call to a logical reasoner is required in order to decide whether or not a set of hypotheses (a subset of the KB) is an explanation

(minimal diagnosis). Incorporating the necessary reasoning costs and assuming consistency a minimal requirement to the correct KB, the finding of the first explanation (minimal diagnosis) is already NP-hard even for propositional KBs [SL89] (since propositional satisfiability checking is NP-complete). The worst case complexity for the debugging of KBs formulated over more expressive logics such as OWL 2 (reasoning is 2-NExpTime-complete [GHM+08, Kaz08]) will be of course even worse. This seems quite discouraging. However, we have shown in our previous works [RSFF13, SFFR12, SFRF14c] that for many real-world KBs interactive KB debugging is feasible in reasonable time, despite high (or intractable) worst case reasoning costs and the intractable complexity of the abduction (i.e. minimal diagnosis finding) problem as such. Hence, the goal of this work is amongst others to present algorithms that work well in many *practical* scenarios.

Assumptions about the Interacting User.   About a user u consulting an (interactive) debugging system, we make the following plausible assumptions:

U1 u is not able to explicitly enumerate a set of logical formulas that express the intended domain that should be modeled in a satisfactory way, i.e. without unwanted entailments or non-fulfilled requirements,

U2 u is able to answer concrete queries about the intended domain that should be modeled, i.e. u can classify a given logical formula (or a conjunction of logical formulas) as a wanted or unwanted proposition in the intended domain (i.e. an entailment or non-entailment of the correct domain model).

The first assumption is obviously justified since otherwise u could have never obtained a faulty KB, i.e. a KB that violates at least one requirement or test case, and there would be no need for u to employ a debugging system.

Regarding the second assumption, the first thing to be noted is that any KB (i.e. any model of the intended domain) either does entail a certain logical formula ax or it does not entail ax. Second, if u is assumed to bring along enough expertise in that domain, u should be able to gauge the truth of (at least) some formulas about that domain, especially if these formulas constitute logical entailments of parts of the specified knowledge in KB so far. We want to emphasize that u is not required to be capable of answering all possible queries (or formulas) about the respective domain since u might always skip a particular query in our system without any noticeable disadvantages. In such a case, the system keeps generating further queries, one at a time (usually the next-best one according to some quality measure for queries), until u is ready to answer it. As the number of possible queries is usually exponential in the number of minimal diagnoses exploited to compute it, there will be plenty of different "surrogate queries" in most scenarios.

A Motivating Example.   To get a more concrete idea of these assumptions, the reader is invited to think about whether the following first-order KB $\mathcal{K}$ is consistent (a similar example is discussed in [HPS09]):

$$\forall X(res(X) \leftrightarrow \forall Y(writes(X, Y) \rightarrow paper(Y))) \tag{1.1}$$

$$\forall X((\exists Y\, writes(X, Y)) \rightarrow res(X)) \tag{1.2}$$

$$\forall X(secr(X) \rightarrow gen(X)) \tag{1.3}$$

$$\forall X(gen(X) \rightarrow \neg res(X)) \tag{1.4}$$

$$secr(pam) \tag{1.5}$$

If we assume that the predicate symbols res, $secr$ and gen stand for 'researcher', 'secretary' and 'general employee', respectively, and the constant pam stands for the person Pam, the KB says the following:

- Formula 1.1: "Somebody is a researcher if and only if everything they write is a paper."

- Formula 1.2: "Everybody who writes something is a researcher."

- Formula 1.3: "Each secretary is a general employee."

- Formula 1.4: "No general employee is a researcher."

- Formula 1.5: "Pam is a secretary."

This KB is indeed inconsistent. The reader might agree that it is not very easy to understand why this is the case. The observations made in [HPS09] concerning a slight modification $\mathcal{K}'$ of the KB $\mathcal{K}$ extracted from a real-world KB confirm this assumption. Compared to $\mathcal{K}$, the KB $\mathcal{K}'$ included only Formulas 1.1-1.3 of $\mathcal{K}$, was formulated in DL (cf. Section 2.2), and used the terms $A, C, \ldots$ instead of $res, paper, \ldots$. Amongst others, this KB $\mathcal{K}'$ was used as a sample KB in a study where participants had to find out whether a concrete given formula is or is not entailed by a concrete given KB. In the case of the KB $\mathcal{K}'$, the assignment (translated to the terminology in our KB $\mathcal{K}$) was to find out whether $\forall X (secr(X) \rightarrow res(X))$ is an entailment of formulas 1.1-1.3. Although $\mathcal{K}'$ contains only three formulas, the result was that even participants with many years of experience in DL, among them also DL reasoner developers, did not realize that this is in fact the case (the reason for this entailment to hold is that formulas 1.1-1.3 imply that $\forall X \, res(X)$ holds).

Since $\forall X \, res(X)$ is also necessary for the inconsistency of $\mathcal{K}$, this suggests that people might also have severe difficulties in comprehending why $\mathcal{K}$ is inconsistent. Once the validity of this entailment is clear, it is relatively straightforward to see that $\mathcal{K}$ cannot have any models. For, $res(pam)$ (due to $\forall X \, res(X)$) and $\neg res(pam)$ (due to formulas 1.3-1.5) are implications of $\mathcal{K}$.

Consequently, we might also assume that even experienced knowledge engineers (not to mention pure domain experts) could end up with a contradictory KB like $\mathcal{K}$, which substantiates our first assumption (U1) about u. Probably, the intention of those people who specified formulas 1.1-1.3 was not that $\forall X \, res(X)$ should be entailed. That is, it might be already a too complex task for many people to (mentally) reason even with such a small KB like this and manually derive implicit knowledge from it.

However, on the other hand, we might well assume u to be able to answer a concrete query about the intended domain they tried to model by $\mathcal{K}$. For instance, one such query could be whether $Q_1 := \{\forall X \, res(X)\}$ is a desired entailment of their model (i.e. "should everybody be a researcher in your intended model of the domain?"). If we assume the (seemingly obvious) case that u negates this query, i.e. asserts that this is an unwanted entailment, then an interactive debugging system (employing a logical reasoner) can derive that at least one of the formulas 1.1 and 1.2 must be faulty. This holds because the only set-minimal explanation in terms of formulas in $\mathcal{K}$ for the entailment $\forall X \, res(X)$ is given by these two formulas. In other words, the set of formulas $\{1.1, 1.2\}$ is the only minimal conflict set in $\mathcal{K}$ given that Q_1 is a negative test case. Hence, the deletion (or suitable modification) of any of these formulas will break this unwanted entailment.

Before it is known that Q_1 must not be entailed by the correct KB, given consistency is the only requirement to the KB postulated by u, the complete KB $\mathcal{K}$ is a minimal conflict set. That is, after the assignment of a (strategically well-chosen) query to the set of positive or, in this case, negative test cases can already shift the focus of potential modifications or deletions to a subset of only two candidate formulas. We would call these two formulas the remaining minimal diagnoses after an answer to the query Q_1 has been submitted.

Initially, there are five minimal diagnoses, each formula in $\mathcal{K}$ is one. The meaning of a diagnosis is that its deletion from $\mathcal{K}$ leads to the fulfillment of all requirements and (so-far-)specified positive and negative test cases. As the reader should be easily able to see, the deletion of any formula from $\mathcal{K}$ yields a consistent KB; e.g. removing formula 1.5 prohibits the entailment $\neg res(pam)$ whereas discarding formula 1.2 prohibits the entailment $res(pam)$. The reader should notice that, as soon as the negative test case Q_1 is known, removing (only) formula 1.5 does not yield a correct KB since $\{1.1, 1.2, 1.3, 1.4\}$ still entails Q_1 which must not be entailed.

A second query to u could be, for example, $Q_2 : \{\exists X((\exists Y \, writes(X, Y)) \wedge \neg res(X))\}$ (i.e. "is there somebody who writes something, but is no researcher?"). Again, it is reasonable to suppose that u might know whether or not this should hold in their intended domain model. The (seemingly obvious) answer in this case would be positive, e.g. because u intends to model students who write homework, exams, etc., but are no researchers. This positive answer leads to the new positive test case Q_2. Adding this positive test case, like a set of new formulas, to the KB $\mathcal{K}$ would result in $\mathcal{K}_{new} := \mathcal{K} \cup Q_2$. The debugging system would then figure out that formula 1.2 is the only minimal conflict set in the KB $\mathcal{K}_{new}$. The reason for this is that the elimination of formula 1.2 breaks the entailment Q_1 (negative test case) and enables the addition of a new desired entailment Q_2 (positive test case) without involving the violation of any requirements (consistency). Therefore, formula 1.2 is the only minimal diagnosis that is still compliant with the new knowledge in terms of $Q_1 = false$ and $Q_2 = true$ obtained.

It is important to notice that the solution KB $\mathcal{K}_{new}$ that is returned to the user as a result of the interactive debugging session includes a new logical formula Q_2 that can be seen as a repair of the deleted formula 1.2. Since the knowledge after the debugging session is that $\neg 1.2 \equiv Q_2$ must be true, this new knowledge is incorporated into the KB $\mathcal{K}_{new}$. This indicates that the fault in KB was simply that the $\neg$ in front of formula 1.2 had been forgotten.

Notice however that the positive test case Q_2 is not added to $\mathcal{K}$ as a usual KB formula, but rather as an *extension* of $\mathcal{K}$ that has already been approved by the user. Should the user at some later point in time commit the same fault again (and explicitly specify some formula x equivalent to formula 1.2), then the interactive debugging system, owing to the positive test case Q_2, would immediately detect a singleton conflict comprising only formula x. As a consequence, each diagnosis considered during this later debugging session would suggest to delete or modify (at least) x.

This scenario should illustrate that, in spite of not being able to specify their domain knowledge in a logically consistent way, the user u might still be able to answer questions about the intended domain, which supports our second assumption made about the user u (the reader might agree that answering Q_1 and Q_2 is much easier than recognizing the entailment $\forall X \, res(X)$ of the KB). In other words, the availability of an (efficient) debugging system could help u debug their KB, without needing to analyze *which* entailments hold or do not hold, *why* certain entailments hold or do not hold or *why* exactly the KB does not meet certain imposed requirements or test cases, by simply answering queries *whether* a certain entailment *should* or *should not* hold. These queries are automatically generated by the system in a way that they focus on the problematic parts of the KB, i.e. the minimal conflict sets, and discriminate between the possible solution candidates, i.e. the minimal diagnoses.

Benefits of the Usage of Conflict Sets.   We want to remark that the usage of minimal conflict sets "naturally" forces the system to take into consideration only the smallest relevant (faulty) parts of the problematic KB. This is owed to the property of minimal conflict sets to abstract from what *all* the reasons for a certain entailment or requirements violation are. Instead, only the "root" (subset-minimal) causes for such violations are examined and no computation time is wasted to extract "purely derived" causes (those which are resolved as a byproduct of fixing all root causes from which it is derived, cf. [Hor11, Kal06]). For example, assuming the debugging scenario involving our example KB consisting only of formulas 1.1-1.4 which is incoherent and a requirements set including coherency. Then, there are two entailments reflecting the incoherency of this KB, first $\forall X \, \neg secr(X)$ and second $\forall X \, \neg gen(X)$ (these entailments hold due to $\forall X \, res(X)$ which follows from formulas 1.1 and 1.2). Of these two, only the second one is a "root" problem; the first one is a "purely derived" problem. That means, the entailment $\forall X \, \neg secr(X)$ only holds due to the presence of the entailment $\forall X \, \neg gen(X)$. So, the cause for $\forall X \, \neg gen(X)$ is given by the set of formulas $\{1.1, 1.2, 1.4\}$ whereas the proper superset $\{1.1, 1.2, 1.3, 1.4\}$ of this set accounts for the entailment $\forall X \, \neg secr(X)$. The exploitation of minimal conflict sets (the only minimal conflict set for this KB is $\{1.1, 1.2, 1.4\}$) ascertains that such "purely derived" causes of requirements or test case violations will not be considered at all.
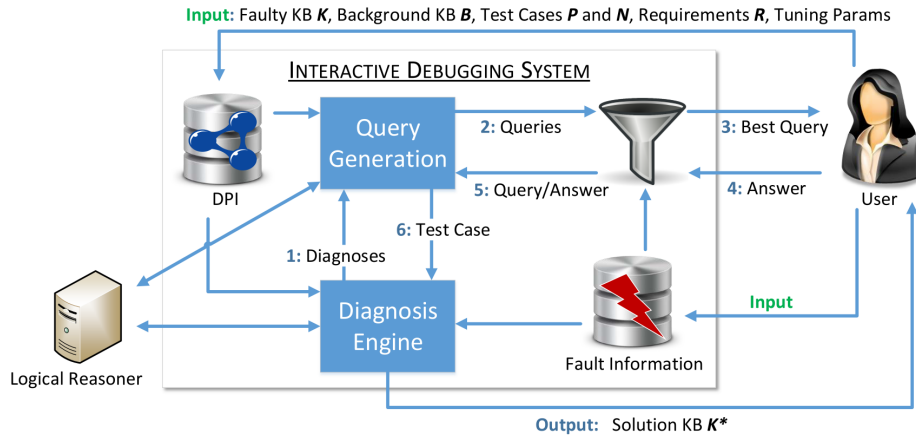
Figure 1.2: The principle of interactive KB debugging.

The Ability to Incorporate Background Knowledge.   Another feature of the approaches described in this work is their ability to incorporate relevant additional information in terms of a background knowledge KB $\mathcal{B}$ (which is regarded to be correct). $\mathcal{B}$ is a (consistent) KB which is usually semantically related with the faulty KB, e.g. $\mathcal{B}$ represents knowledge about the domain modeled by $\mathcal{K}$ that has already been sufficiently endorsed by domain experts. For instance, a doctor who wants to express their knowledge of dermatology in terms of a KB might resort to an approved background KB that specifies the human anatomy. Taking this background information into account puts the problematic KB into some context with existing knowledge and can thereby help a great deal to restrict the search space for solutions of the (interactive) KB debugging problem. This has also been found in [Stu08]. This useful strategy of prior search space restriction is also exploited in the field of ontology matching[6] where automatic systems are employed to generate an alignment, i.e. a set of correspondences between semantically related entities of two different ontologies (KBs). Here, both ontologies are considered correct and diagnoses are only allowed to include elements of the alignment [MST07].

Applying a strategy like that to our example KB given above, supposing that we know that Pam is not a researcher in the world the KB should model, we might specify the background KB $\mathcal{B} := \{\neg res(pam)\}$ prior to starting the interactive debugging session. This would immediately reduce the initial set of possible minimal diagnoses from five (i.e. the entire KB) to two (i.e. the first two formulas 1.1 and 1.2). Reason for this is that the entailment $\forall X \, res(X)$ of formulas 1.1 and 1.2 already conflicts with the background knowledge $\neg res(pam)$.

Outline of an Interactive KB Debugging System.   The schema of an interactive debugging system is pictured by Figure 1.2.[7] As in the case of a non-interactive debugging system (see above), the system receives as input a *diagnosis problem instance (DPI)*. Further on, a range of additional parameters might be provided to the system. These serve as a means to fine-tune the system's behavior in various aspects. Hence, we call these inputs *tuning parameters*. These are (roughly) explained next.

First, some parameters might be specified that take influence on the number of leading diagnoses used for query generation and the necessary computation time invested for leading diagnoses computation. Moreover, some parameter determining the quantity of (pre-)generated queries (of which one is selected to be asked to the user) versus the reaction time (the time it takes the system to compute the next query

---

[6]http://www.ontologymatching.org/

[7]Thanks to Kostyantyn Shchekotykhin for making available to me parts of this diagram.

after the current one has been answered) of the system can be chosen. A further input argument is a query selection measure constituting a notion of query "goodness" that is employed to filter out the "best" query among the set of generated queries. To give the system a criterion specifying when a solution of the interactive KB debugging problem is "good enough", the user is allowed to define a fault tolerance parameter σ. The lower this parameter is chosen, the better the (possibly "approximate") solution that is guaranteed to be found. In case of specifying this parameter to zero, the system will (if feasible) return the "exact" solution of the interactive KB debugging problem. Roughly, the exact solution is given in terms of a solution KB obtained by means of a *single* solution candidate (minimal diagnosis) that is left after a sufficient number of queries have been answered (and added to the test cases). On the contrary, an approximate solution is represented by a solution KB obtained by means of a solution candidate with sufficiently high probability (where "sufficiently high" is determined by σ) at some point where there are still multiple solution candidates available.

Finally, the user may choose between two different modes (*static* or *dynamic*) of determining the leading diagnoses. The *static* diagnosis computation strategy guarantees a constant "convergence" towards the exact solution by "freezing" the set of solution candidates at the very beginning and exploiting answered queries only for the deletion of minimal diagnoses. A possible disadvantage of this approach is the lack of efficient pruning of the used search tree. On the other hand, the *dynamic* method of calculating leading diagnoses has a primary focus on the preservation of a search tree of small size, thereby aiming at being able to solve diagnosis problem instances which are not solvable by the *static* approach due to high time and (more critically) space complexity. To this end, more powerful pruning rules are applied in this case which do not permit the algorithm to consider only a fixed set of solution candidates. Rather, the set of minimal diagnoses and minimal conflict sets are generally variable in this case which means that they are subject to change after assignment of an answered query to the test cases.

Like in the case of a non-interactive debugger, an interactive debugging system requires a sound and complete logical reasoner for deciding consistency (coherency) and calculating logical entailments of a KB formulated over the language $\mathcal{L}$.

The workflow in interactive KB debugging illustrated by Figure 1.2 is the following:

1. A set of leading diagnoses is computed by the diagnosis engine (by means of the fault information, if available) using the logical reasoner and passes it to the query generation module.

2. The query generation module computes a pool of queries exploiting the set of leading diagnoses and delivers it to the query selection module.

3. The query selection module filters out the "best query" (often by means of the fault information, if available) and shows it to the interacting user.

4. The user submits an answer to the query.

5. The query along with the given answer is used to formulate a new test case.

6. This new test case is transferred back to the diagnosis engine and taken into account in prospective iterations. If the stop criterion (as per σ, see above) is not met, another iteration starts at step 1. Otherwise, the solution KB $\mathcal{K}^*$ constructed from the currently most probable minimal diagnosis is output.

Contributions of this Work.   The contributions of this work are the following:

- This work provides a thorough account of the subject and evolves the theory of interactive KB debugging (for monotonic KBs) by presupposing a reader to have only some basic knowledge of logic. Hence, this work addresses newbies as well as people already familiar with related topics. Whereas the comprehensive theoretical considerations might appeal to the more theoretically

oriented readers such as researchers, the precise and exhaustive description of all discussed algorithms might be interesting from the implementation point of view and might serve more practically oriented people such as programmers or engineers as an algorithmic cookbook. Further on, the extensive illustration of the way algorithms work by examples might also serve a merely superficially interested reader to just receive a rough impression of how KBs might be interactively debugged.

- Except for basics in FOL and PL, this work is self-contained and provides all necessary definitions and proofs to make the topic of interactive KB debugging accessible to the reader.

- To the best of our knowledge, this work provides the most comprehensive and detailed introduction to the field of interactive debugging of (monotonic) KBs. Our previous works on the topic [SFFR12, SF10, RSFF13, FS05, SFRF14c] are more application-oriented and thus abstract from some details and omit some of the proofs in favor of comprehensive evaluations of the presented strategies.

- This is the first work that gives formal and precise definitions of problems dealt with in interactive KB debugging and introduces methods that provably solve these problems. We believe that precise problem statements are the very basis for all further scientific investigations in a field. Hence, we hope that this work can "open" the important subject of interactive KB debugging to a broader audience of interested researchers. This can lead to further progress and improvements in debugging techniques which we deem essential in the light of the growing number of intelligent applications incorporating KBs of growing size and complexity (keyword: The Semantic Web [BLHL$^+$01]).

- An in-depth discussion of query computation including computational complexity considerations together with an accentuation of potential ways of improving these methods is given. The investigated methods for query computation have been used also in [SFFR12, RSFF13, SF10, SFRF14c], but have not been addressed in depth in these works.

- We are concerned with the discussion of different ways of exploiting diverse sources of meta information in the KB debugging process from which diagnosis probabilities can be extracted. Our previous works on this topic [SFFR12, RSFF13, SF10, SFRF14c] do not address this matter in a comparable depth.

- We give a formal proof of the soundness of an algorithm QX (based on [Jun04]) for the detection of a minimal conflict set in a KB and we show the correctness (completeness, soundness, optimality) of a hitting set tree algorithm HS (based on [Rei87]) for finding minimal diagnoses in a KB in best-first order (i.e. most probable diagnoses first) which uses QX for conflict set computation only on-demand. We are not aware of any other work that comprises such proofs.

- We establish the theoretical relationship between the widely-used notions of a conflict set and a justification. The former is i.a. used in [dKW87, Rei87, SFFR12, RSFF13] and the latter i.a. in [HPS08, HPS09, HPS10, Hor11, HBP11, HPS12b, SQJH08, Kal06, MS09, SSZ09, NRG12]. As a consequence, empirical results concerning the one might be translated to the other. For instance, since each minimal conflict set is an subset of a justification and there is an efficient (polynomial) method for computing a minimal conflict set given a superset of a minimal conflict set, a result manifesting the efficiency of justification computation for a set of KBs (e.g. [HPS12a]) implies the efficiency of conflict set computation for the same set of KBs. Moreover, we argue that minimal conflict sets are the better choice for our system since these put the focus of the debugger only on the smallest faulty subsets of the KB whereas justifications are better suited in scenarios where exact explanations for the presence of certain entailments are sought.

- Two new algorithms for iterative (leading) diagnosis computation in interactive KB debugging are proposed. One that is guaranteed to reduce the number of remaining solutions after a query is

answered and one that features more powerful pruning techniques than our previously published algorithms [SFFR12, RSFF13] (an evaluation that compares the overall efficiency of our previous algorithms with the ones proposed in this work must still be conducted and is part of our future research).

- We suggest and extensively analyze different methods for the selection of an "optimal" query to ask the user out of a pool of possible queries. We compare a greedy "split-in-half" strategy that proposes queries which eliminate half of the leading diagnoses with a strategy relying on information entropy [Sha48] that chooses the query with highest information gain based on some statistic or (a user's) beliefs about faults in the KB. Comprehensive experiments manifest that only an average guess of the fault information suffices to reduce the query answering effort for the interacting user, often to a significant extent, by means of the latter strategy compared to the former. Moreover, we demonstrate that both methods clearly outperform a random query selection strategy. The latter result witnesses that incorporation of meta (fault) information into the debugging process is in fact reasonable and might relieve the interacting user of a significant proportion of the effort required without taking into account any meta information.

- Addressing the issue of choosing the suitable query selection method for some given fault information, we present a reinforcement learning query selection strategy. For, reliance upon a strategy (e.g. information entropy) that fully exploits and gains from the given fault information can speed up the debugging procedure in the normal case, but can also have a negative impact on the performance in the bad case where the actual solution diagnosis is rated as highly improbable. As an alternative, one might prefer to rely on a tool (e.g. "split-in-half") which does not consider any fault information at all. In this case, however, possibly well-chosen information cannot be exploited, resulting again in inefficient debugging actions.

  Minimal effort for the interacting user can be achieved if both the query selection method is chosen carefully and the provided fault information satisfies some minimum quality requirements. In particular, for deficient fault information and unfavorable strategy for query selection, we observe cases where the overhead in terms of user effort exceeds 2000% (!) in comparison to employing a more favorable query selection strategy. Since, unfortunately, assessment of the fault information is only possible a-poteriori (after the debugging session is finished and the correct solution is known), we devise a learning strategy (RIO) that continuously adapts its behavior depending on the performance achieved and in this vein minimizes the risk of using low-quality fault information.

  This approach makes interactive debugging practical even in scenarios where reliable fault estimates are difficult to obtain. Evaluations provide evidence that for 100% of the cases in the hardest (from the debugging point of view) class of faulty test KBs, RIO performed at least as good as the *best* other strategy and in more than 70% of these cases it even manifested superior behavior to the *best* other strategy. Choosing RIO over other approaches can involve an improvement by the factor of up to 23, meaning that more than 95% of user time and effort might be saved per debugging session.

- We come up with mechanisms for efficiently dealing with KB debugging problems involving high cardinality (minimal) diagnoses. In the standard interactive debugging approach described in the first parts of this work, the computation of queries is based on the generation of the set of *most probable (or minimum cardinality)* leading diagnoses. By this postulation, certain quality guarantees about the output solution can be given. However, we learn that dropping this requirement can bring about substantial savings in terms of time and especially space complexity of interactive debugging, in particular in debugging scenarios where faulty KBs are (partly) generated as a result of the application of automatic systems, e.g. KB (ontology) learning or matching systems [HSNM11, NB12, JMSK09, RP10, JRGZH12, Mei11].
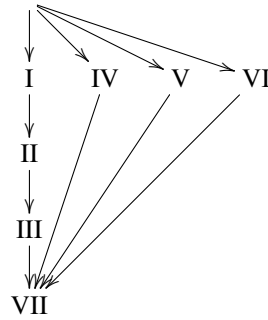
Figure 1.3: Precedence constraints among the parts of this work.

To cope with such situations, we propose to base query computation on *any* set of leading diagnoses using a "direct" method for diagnosis generation. Contrary to the standard method that exploits minimal conflict sets, this approach takes advantage of the duality between minimal diagnoses and minimal conflict sets and employs "inverse" algorithms to those used in the standard approach in order to determine minimal diagnoses directly from the DPI without the indirection via conflict sets.

We study the application of this direct method to high cardinality faults in KBs and find out that the number of required queries per debugging session is hardly affected for cases when the standard approach is also applicable. However, the direct method proves applicable and able to locate the correct solution diagnosis in situations when the standard approach (albeit one that not yet incorporates the powerful search tree pruning techniques introduced in this work) is not due to time or memory issues.

Organization of this Work.   This work is subdivided into seven parts. Figure 1.3 illustrates the precedence constraints among the parts. We want to point out that Parts IV-VI correspond to works that have already been published and are thus self-contained, both from the notation and the content point of view. Parts I-III, on the contrary, are constructive and should thence be read in order.

(Rest of) Part I.   In Chapter 2, besides introducing the notation used in this work, we describe the requirements imposed on logical knowledge representation languages $\mathcal{L}$ that might be used with our approaches. It should be noted that the postulated properties do not restrict the applications of our approaches very much. For instance, these might be employed to resolve over-constrained constraint satisfaction problems (CSPs) or repair faulty KBs in PL, FOL, DL, Datalog or OWL. Since DL provides the logical underpinning of OWL which has recently received increasing attention due to the extensive research in the field of The Semantic Web [BLHL+01], we will also give a short introduction to DL. For, to underline the flexibility of the presented debugging systems in this work, we will illustrate how they work by means of examples involving PL, FOL as well as DL KBs.

In Chapter 3, we first give a formal definition of the *KB debugging* problem and define a diagnosis problem instance (DPI), the input of a KB debugger, and a solution KB, the output of a KB debugger. Further on, we formally characterize a diagnosis and give the notion of KB validity and what it means for a KB to be faulty. We discuss and prove relationships between these notions and specify properties a DPI must satisfy in order to be solvable by a KB debugger.

We motivate why it makes sense to focus on set-minimal diagnoses instead of all diagnoses, i.e. to stick to "The Principle of Parsimony" [Rei87, BATJ91]. This results in the definition of the problem of

parsimonious KB debugging. Then, we prove that solving this problem is equivalent to the computation of a minimal diagnosis. Finally, we explain the benefits of using some background KB in (parsimonious) KB debugging.

In Chapter 4 we describe methods for diagnosis computation. To this end, we first introduce the notion of a (minimal) conflict set, discuss some properties of conflict sets related to the notion of KB validity and give sufficient and necessary criteria for the existence of non-trivial conflict sets w.r.t. a DPI. Subsequently, we derive the relationship between a conflict set and the notion of a justification (a minimal set of formulas necessary for a particular entailment to hold) which is well-known and frequently used, especially in the fields of DL, OWL and The Semantic Web [HPS08, HPS09, HPS10, Hor11, HBP11, HPS12a]. Concretely, we will demonstrate that a minimal conflict set is a subset of a justification for some negative test case or for some inconsistency (entailment *false*) or incoherency (entailment $\forall X_1, \dots, X_k \, \neg p(X_1, \dots, X_k)$ for some predicate symbol p of arity k) of the given KB. Moreover, we will learn that, for the debugging tasks we consider, conflict sets are better suited than justifications.

Having deduced all relevant characteristics of (minimal) conflict sets, we proceed to give a description of a method (QX, Algorithm 1) due to [Jun04] which was originally presented as a method for finding preferred explanations (conflicts) in over-constrained CSPs, but can also be employed for an efficient computation of a minimal conflict set w.r.t. a DPI in KB debugging. We discuss and exemplify this algorithm in detail, prove its correctness as a routine for minimal conflict set computation and give complexity results.

Having at our disposal a proven sound method for generation of a minimal conflict set, we continue with the delineation of a hitting set tree algorithm similar to the one originally presented in [Rei87] which enables the computation of different minimal conflict sets by means of successive calls to QX, each time given an (adequately) modified DPI. In this manner, a hitting set tree can be constructed (breadth-first) which facilitates the computation of minimal diagnoses (minimum cardinality diagnoses first). We prove the correctness (termination, soundness, completeness, minimum-cardinality-first property) of this hitting set tree algorithm coupled with the QX method which serves to solve the problem of parsimonious KB debugging.

In order to be able to incorporate fault information into the diagnoses finding process, we deal with the induction of a probability space over diagnoses in Section 4.6. We discuss several ways of constructing a probability space including different sources of fault information. Hereinafter, we detail how diagnosis probabilities can be determined on the basis of some available fault information and how these can be appropriately updated after new observations (in terms of answered queries) have been made. Furthermore, we outline how fault probabilities can be appropriately incorporated into the hitting set search tree in order to guarantee the discovery of minimal diagnoses in best-first order, i.e. most probable ones first. Then, we prove the correctness (termination, soundness, completeness, best-first property) of this best-first diagnosis finding algorithm for parsimonious KB debugging.

Finally, we describe a non-interactive KB debugging procedure (Algorithm 3) that relies on this best-first diagnosis finding algorithm. Some illustrating examples are provided which at the same time reveal significant shortcomings present in non-interactive KB debugging. This motivates the development of interactive KB debugging algorithms.

Readers not theoretically inclined or non-interested in the technical details might well skip Sections 4.2, 4.4.2, 4.5.2 and 4.6 in Part I.

Part II. In Chapter 6, we first discuss how disadvantages of non-interactive KB debugging procedures can be overcome by allowing a user to take part in the debugging process. Then, we define the problem of *interactive static KB debugging* as well as the problem of *interactive dynamic KB debugging* which "naturally" arise from the fact that the DPI in interactive KB debugging is always renewed after a new test case has been specified (a new query has been answered). The former problem searches for a solution KB *w.r.t. the DPI given as input* such that this solution KB satisfies all test cases added during the debugging

session and there is no other such solution KB. The latter problem searches for a solution KB *w.r.t. the current DPI* (i.e. the input DPI including all new test cases added throughout the debugging session so far) such that there is no other solution KB w.r.t. the current DPI.

Next, in Chapter 7, the central term of a *query* is specified which constitutes the medium for user interaction. Queries are generated from a set of *leading diagnoses* which is characterized thereafter. The set of leading diagnoses is uniquely partitioned into three subsets by each query. The tuple including these subsets is called *q-partition*. Subsequently, the reader is given some explanations how the q-partition can be interpreted, and how it relates to a query. In fact, we will prove that the notion of a q-partition can serve as a criterion for checking whether a set of logical formulas is a query or not. After that, we will learn that a query exists for any set of (at least two) leading diagnoses which grants that the presented algorithms will definitely be able to come up with a query without the need to impose any restrictions on which (minimal) diagnoses are computed by the diagnosis engine in each iteration.

Chapter 8 shows a method for the generation of (a pool of) set-minimal queries (Algorithm 4) aiming at stressing the interacting user as sparsely as possible, features in-depth discussions of this method's properties, proves its correctness, provides complexity results and gives some illustrating examples. Further on, drawbacks of this method are pointed out and possible solutions are discussed.

Subsequently, Chapter 9 deals with the presentation of the central algorithm of this work which implements an interactive KB debugging system (Algorithm 5). First, an overview of the workflow of interactive KB debugging is given, followed by a more comprehensive detailed specification of the algorithm. Some query selection measures are discussed [RSFF13, SFFR12] and optimization versions of the problems of interactive dynamic and static KB debugging are defined where the goal is to obtain the solution to these problems by asking the user a minimal number of queries. Finally, we prove the correctness of the interactive KB debugging algorithm and provide a discussion of its complexity.

Non-theoretically-oriented readers might well skip Sections 8.2, 8.4, 8.5, 8.7 and 9.4 in Part II. Moreover, for the superficially interested reader, it may suffice to concentrate only on Chapter 6 and Sections 7.1, 7.2 and 9.1 in Part II.

Part III.   Here, we go into detail w.r.t. the two strategies for iterative diagnoses computation introduced in Part II that might be plugged into Algorithm 5 to solve either the interactive static or dynamic KB debugging problem.

Chapter 11 describes the *static* method and proves its soundness and completeness w.r.t. the computation of minimal diagnoses w.r.t. the DPI given as an input to the interactive KB debugging algorithm and its optimality w.r.t. the discovery of minimal diagnoses in best-first order (most-probable or minimum cardinality diagnoses first). Incorporation of the *static* method as a routine for leading diagnosis computation into Algorithm 5 provably solves the problem of interactive static KB debugging.

Chapter 12 details the *dynamic* method and proves its soundness and completeness w.r.t. the computation of minimal diagnoses w.r.t. the current DPI and its optimality w.r.t. the discovery of minimal diagnoses in best-first order (most-probable or minimum cardinality diagnoses first). Employing the *dynamic* method as a routine for leading diagnosis computation in Algorithm 5 provably solves the problem of interactive dynamic KB debugging.

The practically oriented reader or the one that is willing to believe that the presented iterative diagnosis computation techniques in fact work as claimed might skip Sections 11.4 as well as 12.4 in Part III.

Part IV.   In this part, we suggest and extensively analyze different methods for the selection of an "optimal" query (see above). The material dealt with in Part IV is based on the publications [SFFR12, SF10] where the former was published in the journal *Web Semantics: Science, Services and Agents on the World Wide Web* and the latter in the *Proceedings of the 9th International Semantic Web Conference (ISWC 2010)*.

Part V. The reinforcement learning query selection strategy (RIO) that makes the presented debugging system robust against the usage of low-quality fault information is presented and thoroughly analyzed in this part which is based on the works [RSFF13, RSFF12, RSFF11, SRF11] published in *Web Reasoning and Rule Systems (RR-2013)*, in the *Proceedings of the 7th International Workshop on Ontology Matching (OM-2012)*, in the *Proceedings of the Joint Workshop on Knowledge Evolution and Ontology Dynamics 2011 (EvoDyn2011)* and in *DX 2011 - 22nd International Workshop on Principles of Diagnosis*, respectively.

Part VI. This part covers the topic of efficiently dealing with KB debugging problems involving high cardinality faults (see above) and relies on material presented in [SFRF14c, SFRF14a, SFRF14b] and published in the *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, in *DX 2014 - 25th International Workshop on Principles of Diagnosis* and in the *Proceedings of the Third International Workshop on Debugging Ontologies and Ontology Mappings (WoDOOM14)*, respectively.[8]

Part VII. To round this work off, we provide a discussion of related work in Chapter 32,[9] summarize the contributions of this work in Chapter 33 and deal with our future work topics in Chapter 34.

---

[8]We are glad to report that the publication [SFRF14a] was awarded the Best Paper Award at the DX Workshop that took place in Graz, Austria, in September 2014 (see http://dx-2014.ist.tugraz.at).

[9]Note that related work specific to topics addressed in Parts IV-VI is separately treated in these parts.

Chapter 2

Preliminaries

2.1 Assumptions

The techniques described in this work are applicable for any logical knowledge representation formalism $\mathcal{L}$ for which the entailment relation is

1. *monotonic:* is given when adding a new logical formula to a KB $\mathcal{K}_\mathcal{L}$ cannot invalidate any entailments of the KB, i.e. $\mathcal{K}_\mathcal{L} \models \alpha_\mathcal{L}$ implies that $\mathcal{K}_\mathcal{L} \cup \{\beta_\mathcal{L}\} \models \alpha_\mathcal{L}$,

2. *idempotent:* is given when adding implicit knowledge explicitly to a KB $\mathcal{K}_\mathcal{L}$ does not yield new entailments of the KB, i.e. $\mathcal{K}_\mathcal{L} \models \alpha_\mathcal{L}$ and $\mathcal{K}_\mathcal{L} \cup \{\alpha_\mathcal{L}\} \models \beta_\mathcal{L}$ implies $\mathcal{K}_\mathcal{L} \models \beta_\mathcal{L}$ and

3. *extensive:* is given when each logical formula entails itself, i.e. $\{\alpha_\mathcal{L}\} \models \alpha_\mathcal{L}$ for all $\alpha_\mathcal{L}$,

and for which

4. reasoning procedures for *deciding consistency* and *calculating logical entailments* of a KB are available,

where $\alpha_\mathcal{L}, \beta_\mathcal{L}$ are logical formulas and $\mathcal{K}_\mathcal{L}$ is a set $\left\{ ax_\mathcal{L}^{(1)}, \ldots, ax_\mathcal{L}^{(n)} \right\}$ of logical formulas formulated over the language $\mathcal{L}$. $\mathcal{K}_\mathcal{L}$ is to be understood as the conjunction $\bigwedge_{i=1}^{n} ax_\mathcal{L}^{(i)}$. Notice that the elements of a KB are called quite differently in literature. Possible denotations are logical formula (e.g. [KK06]), well-formed formula (e.g. [CL73]), (logical) sentence or axiom (e.g. [RN10]) and axiom (in most of the description logic literature, e.g. [BCM$^+$07]). We will mainly stick to the term *formula* (sometimes *axiom*) to refer to the elements of a KB. As the logic will be clear from the context in the sequel, we will omit the index $\mathcal{L}$ when referring to formulas or KBs over $\mathcal{L}$ throughout the rest of this work.

2.2 Considered Logics

To underline the general character of this work, we will illustrate our approaches using example diagnosis problem instances expressed in different logical languages. In this section we give notational remarks concerning these different logics used, namely propositional logic (PL), first-order logic (FOL) as well as description logic (DL). Whereas we assume the reader to be familiar with FOL and PL (a good introduction to PL and FOL can be found in [CL73]), we will give a short introduction to DL.

Remark 2.1 It is important to notice that the usage of DL as well as FOL examples throughout this work should *not* suggest that the Properties 1 – 4 stated above are satisfied for any DL or FOL language

$\mathcal{L}$. In fact, it is well-known by the theorems of Church and Turing (cf. [Men09]; the original works are [Chu36, Tur37]) that FOL is not decidable in general, i.e. Property 4 above is not met. Also in the case of DL, which subsumes a range of different logical languages featuring different expressivity and thus different computational complexity of reasoning procedures, there are languages which are undecidable. For instance, a DL language allowing the formalism of equality role-value-maps which facilitates the expression of concepts like "persons whose co-workers coincide with their relatives" can be proven undecidable [BCM⁺07, SS89].

Property 4 is satisfied, for example, for the DL language $\mathcal{SROIQ}$ which is the logical underpinning of OWL 2 [GHM⁺08]. However, the complexity (2-NExpTime-complete [Kaz08]) of logical reasoning is intractable in the worst case for this language which implies the intractability of our methods in the worst case. Nevertheless, other DL languages applied with similar systems as those described in this paper have been showing reasonable performance [SFRF14c, RSFF13, SFFR12]. Also from the theoretical point of view, there are DL languages that allow for efficient reasoning. One example is the OWL 2 EL profile which enables polynomial time reasoning [BBL05]. For this language, the efficient reasoning service ELK has been presented by [KKS14]. For FOL, datalog is an example of a decidable sublanguage where reasoning is efficient [RN10]. Further, restricted sublanguages of FOL can often be translated to some DL language wherefore DL positive results concerning the decidability of reasoning as well as complexity results can be adopted for these restricted FOL languages [BCM⁺07, chapter 4] [Bor96].

Moreover, we want to point out that the practical efficiency of our systems depends strongly on the practical performance (which might be by far better than suggested by the worst case reasoning complexities) of the reasoning services called by our algorithms since the reasoning services are used as a black-box (as mentioned in Chapter 1). Possible strategies for improving the reasoning efficiency in the black-box setting are briefly discussed in Chapter 34. $\square$

Ontologies and The Semantic Web

Ontologies are KBs that formally and explicitly represent common knowledge about a domain in the form of individuals, concepts (set of individuals) and roles (binary relationships between individuals). As, in the last decade, extensive research has been done in the area of The Semantic Web [BLHL⁺01] making (automatic) ontology development tools and reasoning services more efficient, ontology engineering for the Semantic Web is on the upswing. The Semantic Web aims at the enrichment of unstructured information on the web by semantic meta data which should facilitate the usage of the web as structured database of knowledge of all kinds where computers are able to "understand" this structured data, establish relationships between different data sources, combine information from different data sources and (most essentially) derive new (implicit) knowledge from the structured data. At this, ontologies are the key to a common vocabulary used for the semantic meta data. Ontologies are employed to precisely define the meaning of different terms, state relationships between different terms and to introduce new terms by means of already specified ones.

The constantly increasing number of people creating ontologies of increasing size (examples were given in Chapter 1) results in more and more (faulty) ontologies which constitute useful application scenarios and test cases for our approaches. For that reason, we also want to use ontology engineering for The Semantic Web as a concrete use case for the presented work. The standard knowledge representation formalism for ontologies is OWL 2 [MPSP09, GHM⁺08] which relies on DL. A short introduction to DL is given next.

Description Logic

Description Logic (DL) [BCM⁺07] is a family of knowledge representation languages with a formal logic-based semantics that are designed to represent knowledge about a domain in form of concept de-

scriptions. The *syntax* of a description language $\mathcal{L}$ is defined by its signature and a set of constructors. The *signature* of $\mathcal{L}$ corresponds to the union of possibly disjoint sets N_C, N_R and N_I, where N_C contains all concept names (unary predicates), N_R comprises all role names (binary predicates) and N_I is the set of all individuals (constants) in $\mathcal{L}$. Each concept and role description can be either atomic or complex. The latter ones are composed using constructors defined in the particular language $\mathcal{L}$. A typical set of DL *constructors* for complex concepts includes conjunction $A \sqcap B$, disjunction $A \sqcup B$, negation $\neg A$, existential $\exists r.A$ and value $\forall r.A$ restrictions, where A, B are concept descriptions and $r \in N_R$.

Axioms are statements of knowledge that must be true in a domain. An *ontology* $\mathcal{K}$ is defined as a tuple $(\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ (*TBox*) is a set of terminological axioms and $\mathcal{A}$ (*ABox*) a set of assertional axioms. Each TBox axiom is expressed by a general concept inclusion $A \sqsubseteq B$, a form of logical implication, or by a definition $A \equiv B$, a kind of logical equivalence, where A and B are concept descriptions or role descriptions. ABox axioms are used to assert properties of individuals in terms of the vocabulary defined in the TBox, e.g. concept $A(x)$ or role $r(x, y)$ assertions, where A is a concept description, r a role description, and $x, y \in N_I$.

The semantics of a description language is given in terms of *interpretations* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a non-empty domain $\Delta^{\mathcal{I}}$ and a function $\cdot^{\mathcal{I}}$ that assigns to every atomic concept $A \in N_C$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, to every atomic role $r \in N_R$ a set $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and to every individual $x \in N_I$ some value $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The interpretation function is extended to complex concept descriptions by the following inductive definitions:

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$$
$$\bot^{\mathcal{I}} = \emptyset$$
$$(A \sqcap B)^{\mathcal{I}} = A^{\mathcal{I}} \cap B^{\mathcal{I}}$$
$$(A \sqcup B)^{\mathcal{I}} = A^{\mathcal{I}} \cup B^{\mathcal{I}}$$
$$(\neg A)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$$
$$(\exists r.A)^{\mathcal{I}} = \left\{ x \in \Delta^{\mathcal{I}} \mid \exists y. (x, y) \in r^{\mathcal{I}} \wedge y \in A^{\mathcal{I}} \right\}$$
$$(\forall r.A)^{\mathcal{I}} = \left\{ x \in \Delta^{\mathcal{I}} \mid \forall y. (x, y) \in r^{\mathcal{I}} \rightarrow y \in A^{\mathcal{I}} \right\}$$

where $\top$ and $\bot$ are predefined concepts; the former is the universal concept and the latter the bottom concept.

The semantics of axioms is defined as follows for (1) TBox and (2) ABox axioms: (1) Interpretation $\mathcal{I}$ satisfies $A \sqsubseteq B$ iff $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ and it satisfies $A \equiv B$ iff $A^{\mathcal{I}} = B^{\mathcal{I}}$. (2) $A(x)$ is satisfied by $\mathcal{I}$ iff $x^{\mathcal{I}} \in A^{\mathcal{I}}$ and $r(x, y)$ is satisfied iff $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is a *model* of $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ iff it satisfies all TBox axioms in $\mathcal{T}$ and all ABox axioms in $\mathcal{A}$. An *ontology* $\mathcal{K}$ *is consistent* iff it has a model. A *concept* A (*role* r) *is satisfiable* w.r.t $\mathcal{K}$ iff there is a model $\mathcal{I}$ of $\mathcal{K}$ with $A^{\mathcal{I}} \neq \emptyset$ ($r^{\mathcal{I}} \neq \emptyset$). An *ontology* $\mathcal{K}$ *is coherent* iff all concepts and roles occurring in $\mathcal{K}$ are satisfiable. An *axiom* α *is entailed* by $\mathcal{K}$ iff α is true in all models $\mathcal{I}$ of $\mathcal{K}$. For a set of axioms X we write $\mathcal{K} \models X$ as a shorthand for $\mathcal{K} \models \alpha$ for all $\alpha \in X$.

Usually description logic systems provide sound and complete reasoning services to their users. Besides *verification of coherency and consistency of* $\mathcal{K}$ and *satisfiability checking of concepts*, reasoner tasks include classification and realization. *Classification* determines, for each concept name A occurring in $\mathcal{K}$, most specific (general) concepts that subsume (are subsumed by) A. A concept A subsumes (is subsumed by) a concept B iff $\mathcal{K} \models B \sqsubseteq A$ ($\mathcal{K} \models A \sqsubseteq B$). Classification is employed to build a taxonomy of concepts in $\mathcal{K}$. *Realization*, given an individual name x occurring in $\mathcal{K}$ and a given set of concepts in $\mathcal{K}$ (usually all concepts in $\mathcal{K}$), computes the most specific concepts $A_1, \ldots, A_n$ from the set such that $\mathcal{K} \models A_i(x)$ for all $i = 1, \ldots, n$. The most specific concepts are those that are minimal w.r.t. the subsumption ordering $\sqsubseteq$.

Example 2.1 The example KB given in the Introduction (Chapter 1) can be equivalently represented in DL (cf. Remark 2.1) as follows:

$$Res \equiv \forall writes.Paper \tag{2.1}$$

$$\exists writes.\top \sqsubseteq Res \tag{2.2}$$

$$Secr \sqsubseteq Gen \tag{2.3}$$

$$Gen \sqsubseteq \neg Res \tag{2.4}$$

$$Secr(pam) \tag{2.5}$$

where Res is the concept symbol with equivalent meaning as the predicate symbol res, the role symbol $writes$ corresponds to the equally named binary predicate, $Paper$ to $paper$, and so on. Notice that axiom 2.2 states that the domain of $writes$ is Res. □

2.3 Notational Remarks[10]

General Notational Conventions. Throughout this work, the nomenclature given by Table 2.1 is used (many of the designators in the table will be explained later in this work). We will mainly refer to an ontology by the term *KB*.

In order to make a clear distinction between scalars and functions, we denote all scalars g by g and all functions g by $g()$. If an ordered list occurs in a set operation, then this list is interpreted as a (non-ordered) set. For example, let $L := [1, 3, 4, 2]$ be an ordered list; then $L \cap \{1, 2, 3\}$ yields the *set* $\{1, 2, 3\}$.

Notational Convention for PL (cf. [RN10]). We use uppercase letters $A, B, \ldots$ to denote atoms and the standard logical connectives to build PL formulas from atoms. The operator precedence we use is $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, from highest to lowest. Given a PL KB $\mathcal{K}$ and a PL formula ax, we call $\widetilde{\mathcal{K}}$ and $\widetilde{ax}$ the *signature of* $\mathcal{K}$ and the *signature of* ax, respectively. The former comprises all atoms occurring in $\mathcal{K}$ and the latter all atoms occurring in ax.

Notational Convention for FOL (cf. [CGT89]). Variables are denoted by uppercase letters; constants and predicate symbols are denoted by strings beginning with a lowercase letter[11]. Recalling the example KB given in Chapter 1, X, Y are variables, pam is a constant and $res, writes, paper, secr$ and gen are predicate symbols. FOL formulas are built from the standard logical connectives described for PL above. The operator precedence we use for FOL formulas is the same as stated above[12]. The precedence of quantifiers $\forall, \exists$ is such that a quantifier outside of any parenthesized expression holds over everything to the right of it; if occurring in a parenthesized expression, a quantifier holds over everything to the right of it within this expression. For example, $\forall X\, prof(X) \rightarrow \exists Y\, secr(Y)$ is equivalent to $(\forall X (prof(X) \rightarrow (\exists Y (secr(Y)))))$ (i.e. "for each professor there is at least one secretary") and not to $(\forall X\, prof(X)) \rightarrow \exists Y\, secr(Y)$ (i.e. "if everybody is a professor, then there is at least one secretary").

Given a FOL KB $\mathcal{K}$ and a FOL formula ax, we call $\widetilde{\mathcal{K}}$ and $\widetilde{ax}$ the *signature of* $\mathcal{K}$ and the *signature of* ax, respectively. The former comprises all predicate, function and constant symbols occurring in $\mathcal{K}$ and the latter all predicate, function and constant symbols occurring in ax. The signature of the example KB given in Chapter 1 is $\{res, writes, paper, secr, gen, pam\}$ and the signature of formula 1.2 of this KB is $\{writes, res\}$.

---

[10]These conventions apply to Parts I-III and Part VII. Each of the Parts IV-VI is self-contained w.r.t. the used notation.

[11]We do not use any function symbols throughout this work.

[12]We do not use equality $=$ in FOL formulas throughout this work.

Remark 2.2   By analogy with the definition of coherency in DL (see Section 2.2), we call a FOL KB $\mathcal{K}$ *incoherent* iff $\mathcal{K} \models \forall X_1, \ldots, X_k \, \neg p(X_1, \ldots, X_k)$ for some k-place predicate symbol p in the signature of $\mathcal{K}$ where $k \geq 1$. $\square$

Remark 2.3   We want to point out that whenever we will speak of *entailment computation* we address the invocation of a sound reasoning service that is guaranteed to terminate after *finite* execution time and returns a *finite* number of entailments for any KB given as input (cf. Remark 2.1). Similarly, when we say that *all entailments* of a KB are computed, we always refer to a *finite* set of entailments of certain types output by such a reasoning service. Examples of such entailment types regarding DL are the (a) classification and (b) realization entailments, by which we mean (a) all the subsumption relationships between concept names appearing in the KB, i.e. entailments of the form $C_1 \sqsubseteq C_2$ for concept names $C_1, C_2 \in \widetilde{\mathcal{K}}$ and (b) all the concept names instantiated by a given individual for all individuals appearing in the KB, i.e. entailments of the form $C(a)$ for concepts names $C \in \widetilde{\mathcal{K}}$ and individual names $a \in \widetilde{\mathcal{K}}$. $\square$

Symbol	Meaning
2^X	the powerset of X where X is a set
U_X	the union of all elements in X where X is a set of sets
$\mathcal{L}$	a (monotonic, idempotent, extensive) logical knowledge representation language
$\mathcal{K}_{(i)}$	a (faulty) KB (optionally with an index)
$ax_{(i)}$	a formula in a KB (an axiom in an ontology)
$\mathcal{B}_{(i)}$	a (correct) background KB (optionally with an index)
P	the set of positive test cases (each test case is a set of logical formulas)
$p_{(i)}$	a positive test case (optionally with an index)
N	the set of negative test cases (each test case is a set of logical formulas)
$n_{(i)}$	a negative test case (optionally with an index)
R	the set of requirements to the correct KB
$\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$	a diagnosis problem instance (DPI)
$\mathbf{aD}_{DPI}$	the set of all diagnoses w.r.t. the DPI DPI
$\mathbf{mD}_{DPI}$	the set of minimal diagnoses w.r.t. the DPI DPI
$\mathcal{D}_{(i)}$	a (minimal) diagnosis (optionally with an index)
$\mathcal{D}_t$	the true diagnosis
$\mathbf{aC}_{DPI}$	the set of all conflict sets w.r.t. the DPI DPI
$\mathbf{mC}_{DPI}$	the set of minimal conflict sets w.r.t. the DPI DPI
$\mathcal{C}_{(i)}$	a (minimal) conflict set (optionally with an index)
$\mathbf{Q}$	an ordered queue of open nodes in a hitting set tree algorithm
$\mathsf{n}_{(i)}, \mathsf{nd}_{(i)}, \mathsf{node}_{(i)}$	nodes in a hitting set tree algorithm (optionally with an index)
$[a_1, \ldots, a_n]$	context-dependent (will be clear from the context): (1) an ordered list of the elements $a_1, \ldots, a_n$ or (2) a (non-ordered) minimal diagnosis comprising formulas $a_1, \ldots, a_n$
$\langle a_1, \ldots, a_n \rangle$	context-dependent (will be clear from the context): (1) a tuple of elements $a_1, \ldots, a_n$ or (2) a (non-ordered) minimal conflict set comprising formulas $a_1, \ldots, a_n$
u	the user interacting with the debugging system
$u()$	the (user) function that maps queries to answers
$Q_{(i)}$	a query (optionally with an index)
$\mathbf{Q}_{\mathbf{D}, DPI}$	the set of all queries w.r.t. the leading diagnoses $\mathbf{D}$ and the DPI DPI
$\mathfrak{P}(Q)$	the q-partition of the query Q (abbreviated form)
$\langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q) \rangle$	the q-partition of the query Q (written-out form)
$\mathbf{EX}(\mathcal{D})_{DPI}$	the set of all extensions w.r.t. a diagnosis $\mathcal{D}$ and a DPI DPI
$\mathbf{Sol}_{DPI}$	the set of all solution KBs w.r.t. the DPI DPI
$\mathbf{Sol}_{DPI}^{\max}$	the set of all maximal solution KBs w.r.t. the DPI DPI

Table 2.1: Symbols and abbreviations used throughout this work (cf. footnote 10).

Chapter 3

Knowledge Base Debugging

KB debugging can be seen as a test-driven procedure comparable to test-driven software development and debugging, where test cases are specified to restrict the possible faults until the user detects the actual fault manually or there is only one (highly probable) fault remaining which is in line with the specified test cases. In this chapter, we want to study the theory of (non-interactive) KB debugging, present and discuss mechanisms that can be employed for the debugging of KBs and reveal drawbacks of such systems. In (non-interactive) KB debugging we assume *test cases fixed during the debugging procedure*. That is, a user might specify a set of test cases offline, run a debugging system and investigate the output solution(s). In case no satisfactory solution has been returned, some additional test cases might be defined offline before the debugger might be invoked again.

The inputs to a KB debugging problem can be characterized as follows: Given is a KB $\mathcal{K}$ and a KB $\mathcal{B}$ (background knowledge), both formulated over some logic $\mathcal{L}$ complying with the conditions 1 – 4 given in Chapter 2. All formulas in $\mathcal{B}$ are considered to be correct and all formulas in $\mathcal{K}$ are considered potentially faulty. $\mathcal{K} \cup \mathcal{B}$ does not meet postulated requirements R where $\{\text{consistency}\} \subseteq R \subseteq \{\text{coherency, consistency}\}$ or does not feature desired semantic properties, called test cases.[13] Positive test cases (aggregated in the set P) correspond to desired entailments and negative test cases (N) represent undesired entailments of the correct (repaired) KB (along with the background KB $\mathcal{B}$). Each test case $p \in P$ and $n \in N$ is *a set of* logical formulas over $\mathcal{L}$. The meaning of a positive test case $p \in P$ is that the correct KB integrated with $\mathcal{B}$ must entail each formula (or the conjunction of formulas) in p, whereas a negative test case $n \in N$ signalizes that some formula (or the conjunction of formulas) in n must not be entailed by the correct KB integrated with $\mathcal{B}$.

Remark 3.1 In the sequel, we will write $\mathcal{K} \models X$ for some set of formulas X to denote that $\mathcal{K} \models ax$ for all $ax \in X$ and $\mathcal{K} \not\models X$ to state that $\mathcal{K} \not\models ax$ for some $ax \in X$. □

The described inputs to the KB debugging problem are captured by the notion of a diagnosis problem instance:

Definition 3.1 (Diagnosis Problem Instance). *Let*

- $\mathcal{K}$ *be a KB over* $\mathcal{L}$,

- P, N *sets including sets of formulas over* $\mathcal{L}$,

- $\{consistency\} \subseteq R \subseteq \{coherency, consistency\}$,

---

[13]We assume consistency a minimal requirement to a solution KB provided by a debugging system, as inconsistency makes a KB completely useless from the semantic point of view.

- *$\mathcal{B}$ be a KB over $\mathcal{L}$ such that $\mathcal{K} \cap \mathcal{B} = \emptyset$ and $\mathcal{B}$ satisfies all requirements $r \in R$,*

- *the cardinality of all sets $\mathcal{K}$, $\mathcal{B}$, P, N be finite.*

Then we call the tuple $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ a diagnosis problem instance (DPI) *over $\mathcal{L}$.*[14]

Note that, for now, we do not make any assumptions about the contents of the sets $\mathcal{K}$, $\mathcal{B}$, P and N that go beyond Definition 3.1. So, it might be well the case, for example, to specify a DPI according to Definition 3.1 for which there are no solutions or for which only trivial solutions exist. Later on, we will discuss properties a DPI must fulfill to guarantee existence of solutions for it.

We define a solution KB for a DPI as follows:

Definition 3.2 (Solution KB)**.** *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI. Then a KB $\mathcal{K}^*$ is called* solution KB *w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, written as $\mathcal{K}^* \in \mathbf{Sol}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, iff all the following conditions hold:*

$$\forall\, r \in R \quad : \quad \mathcal{K}^* \cup \mathcal{B} \text{ fulfills } r \tag{3.1}$$

$$\forall\, p \in P \quad : \quad \mathcal{K}^* \cup \mathcal{B} \models p \tag{3.2}$$

$$\forall\, n \in N \quad : \quad \mathcal{K}^* \cup \mathcal{B} \not\models n. \tag{3.3}$$

A solution KB $\mathcal{K}^*$ w.r.t. a DPI is called maximal*, written as $\mathcal{K}^* \in \mathbf{Sol}^{\max}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, iff there is no solution KB $\mathcal{K}'$ such that $\mathcal{K}' \cap \mathcal{K} \supset \mathcal{K}^* \cap \mathcal{K}$.*

Now, the problem of KB debugging can be formalized:

Problem Definition 3.1 (KB Debugging)**.** *Given a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, find a solution KB w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Note that basically any KB $\mathcal{K}^*$ that meets conditions (3.1) - (3.3) is a solution KB in the sense of Definition 3.2. Hence, $\mathcal{K}^*$ does not even need to have a non-empty intersection with $\mathcal{K}$. Only the postulation of maximality of a solution KB (as detailed later in Section 3.1) establishes a relationship to the given KB $\mathcal{K}$.

Remark 3.2 Let $\mathcal{K}' := \mathcal{K} \cup \mathcal{B} \cup U_P$. Then, conditions (3.1) - (3.3) can be reduced to conditions (3.2) and (3.3) if

- $N := N \cup \{\{false\}\}$ given $R = \{$consistency$\}$ or

- $N := N \cup \{\{\forall X_1, \ldots, X_k\, p(X_1, \ldots, X_k) \to false\} \mid p$ is k-place predicate symbol in $\widetilde{\mathcal{K}'}, k \geq 1\} \cup \{\{false\}\}$ in case $R = \{$consistency, coherency$\}$.

This holds because a KB $\mathcal{K}$ is inconsistent iff $\mathcal{K} \models \{false\}$ and $\mathcal{K}$ is incoherent iff some predicate symbol in $\mathcal{K}'$ must be *false* for any instantiation. Notice that the latter must hold for all predicate symbols in $\mathcal{K}'$ and not only in $\mathcal{K}$ (see Example 3.1). For PL and DL, the definitions of N are analogous (cf. Chapter 2), but for PL coherency is not defined wherefore only the first bullet is relevant for PL. In what follows we will stick to the more explicit characterization of a solution KB given by Definition 3.2. $\square$

---

[14]In the following we will often call a DPI over $\mathcal{L}$ simply a DPI for brevity and since the concrete logic will not be relevant in our theoretical analyses as long as it is compliant with the conditions $1 - 4$ given in Chapter 2. Nevertheless we will mean exactly the logic over which a particular DPI is defined when we use the designator $\mathcal{L}$.

Example 3.1   Let a DL DPI be defined as

$$\mathcal{K} := \{B \sqsubseteq C\}$$
$$\mathcal{B} := \{A \sqsubseteq B, C \sqsubseteq \neg A\}$$
$$P := \emptyset$$
$$N := \emptyset$$
$$R := \{\text{coherency, consistency}\}$$

Then, $\widetilde{\mathcal{K}} = \{B, C\}$, but there is some concept $A \notin \widetilde{\mathcal{K}}$, but $A \in \widetilde{\mathcal{K}'}$, which is unsatisfiable w.r.t. $\mathcal{K} \cup \mathcal{B}$. Since we want a solution KB *integrated with* $\mathcal{B}$ to meet the conditions (3.1) - (3.3), $\mathcal{K}$ is not a solution KB w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ despite the fact that it is perfectly consistent and coherent as an isolated KB.   □

Whereas the definition of a solution KB refers to the desired properties of the output of a KB debugging system, the following definition can be seen as a characterization of KBs provided as an input to a KB debugger. If a KB is valid w.r.t. the background knowledge, the requirements and the test cases, then finding a solution KB w.r.t. the DPI is trivial. Otherwise, obtaining a solution KB from it involves modification of the input KB and subsequent addition of suitable formulas. Usually, the KB $\mathcal{K}$ part of the DPI given as an input to a debugger is assumed to be invalid w.r.t. this DPI.

Definition 3.3 (Valid KB).   *Let* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *be a DPI. Then, we say that a KB* $\mathcal{K}'$ *is* valid *w.r.t.* $\langle \cdot, \mathcal{B}, P, N \rangle_R$ *iff* $\mathcal{K}' \cup \mathcal{B} \cup U_P$ *does not violate any* $r \in R$ *and does not entail any* $n \in N$. *A KB is said to be* invalid *(or faulty) w.r.t.* $\langle \cdot, \mathcal{B}, P, N \rangle_R$ *iff it is not valid w.r.t.* $\langle \cdot, \mathcal{B}, P, N \rangle_R$.[15]

Intuitively, if a KB $\mathcal{K}$ is faulty w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, then there is at least one incorrect formula in $\mathcal{K}$ that needs to be corrected or deleted; if a KB $\mathcal{K}$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, a solution KB can be *directly* obtained by simply extending $\mathcal{K}$ by the set U_P of all sentences comprised in positive test cases. Note, however, that $\mathcal{K}$ being valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$ does not necessarily mean that $\mathcal{K} \cup \mathcal{B}$ entails any $p \in P$.

Proposition 3.1.   *Let* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *be a DPI. Then,* $\mathcal{K}' \cup U_P \in \mathbf{Sol}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ *iff* $\mathcal{K}'$ *is valid w.r.t.* $\langle \cdot, \mathcal{B}, P, N \rangle_R$.

Proof. "⇒": If $\mathcal{K}' \cup U_P$ is a solution KB, then $\mathcal{K}' \cup U_P \cup \mathcal{B}$ meets all $r \in R$ as per condition (3.1) and does not entail any $n \in N$ as per condition (3.3). Hence, $\mathcal{K}'$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$.

"⇐": If $\mathcal{K}'$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, then $(\mathcal{K}' \cup U_P) \cup \mathcal{B}$ meets all $r \in R$, i.e. meets condition (3.1). Moreover, $(\mathcal{K}' \cup U_P) \cup \mathcal{B} \not\models n$ for all $n \in N$, i.e. $(\mathcal{K}' \cup U_P) \cup \mathcal{B}$ meets condition (3.3). By extensiveness of the used language $\mathcal{L}$, $(\mathcal{K}' \cup U_P) \cup \mathcal{B} \models p$ for all $p \in P$, i.e. condition (3.2) is fulfilled by $(\mathcal{K}' \cup U_P) \cup \mathcal{B}$. Thus, $\mathcal{K}' \cup U_P$ is a solution KB.   □

Definition 3.4 (Extension).   *Let* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *be a DPI over* $\mathcal{L}$ *and* $\mathcal{K}' \subseteq \mathcal{K}$. *A set of formulas* $\mathcal{E}$ *over* $\mathcal{L}$ *is called an* extension *w.r.t.* $\mathcal{K}'$ *and* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, *written as* $\mathcal{E} \in \mathbf{EX}(\mathcal{K}')_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, *iff* $(\mathcal{K} \setminus \mathcal{K}') \cup \mathcal{E}$ *is a solution KB w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Definition 3.5 (Diagnosis).   *Let* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *be a DPI. A set of formulas* $\mathcal{D} \subseteq \mathcal{K}$ *is called a* diagnosis *w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, *written as* $\mathcal{D} \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, *iff there exists some* $\mathcal{E} \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, *i.e.* $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E}$ *is a solution KB w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

A diagnosis $\mathcal{D}$ *w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *is* minimal, *written as* $\mathcal{D} \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, *iff there is no* $\mathcal{D}' \subset \mathcal{D}$ *such that* $\mathcal{D}'$ *is a diagnosis w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. *A diagnosis* $\mathcal{D}$ *w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *is a* minimum cardinality *diagnosis w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *iff there is no diagnosis* $\mathcal{D}'$ *w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *such that* $|\mathcal{D}'| < |\mathcal{D}|$.

---

[15] It would be more precise to call a KB valid *w.r.t. the elements* $\mathcal{B}$, P, N, R of a DPI. Though, for brevity, we stick to the presented notation where the dot $\cdot$ in $\langle \cdot, \mathcal{B}, P, N \rangle_R$ signalizes the irrelevance of the first element $\mathcal{K}$ of a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ for determining validity of a KB $\mathcal{K}'$ w.r.t. this DPI.

Proposition 3.2. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI. Then, $\mathcal{D} \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ iff $\mathcal{K} \setminus \mathcal{D}$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$.*

Proof. "⇒": If $\mathcal{D}$ is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, there is some extension $\mathcal{E}$ w.r.t. $\mathcal{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ which implies that $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E}$ is a solution KB w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Now, assume that $\mathcal{K} \setminus \mathcal{D}$ is not valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. By Proposition 3.1, this means that $(\mathcal{K} \setminus \mathcal{D}) \cup U_P$ is not a solution KB. Hence, $(\mathcal{K} \setminus \mathcal{D}) \cup U_P \cup \mathcal{B}$ violates some $r \in R$ or entails some $n \in N$. As $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E}$ is a solution KB, we have that $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E} \cup \mathcal{B} \models p$ for all $p \in P$. So, by idempotency of $\mathcal{L}$, $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E} \cup \mathcal{B} \equiv (\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E} \cup \mathcal{B} \cup U_P \supseteq (\mathcal{K} \setminus \mathcal{D}) \cup U_P \cup \mathcal{B}$ which violates some $r \in R$ or entails some $n \in N$. By monotonicity of $\mathcal{L}$, $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E} \cup \mathcal{B}$ also violates some $r \in R$ or entails some $n \in N$ whereby $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E}$ is not a solution KB which is a contradiction.

"⇐": If $\mathcal{K} \setminus \mathcal{D}$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, then $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup U_P$ does not violate any $r \in R$ and does not entail any $n \in N$. Since $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup U_P$ also entails each positive test case $p \in P$ by extensiveness of $\mathcal{L}$, we can conclude that $(\mathcal{K} \setminus \mathcal{D}) \cup U_P$ is a solution KB. By Definition 3.4, $U_P \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ and thus $\mathcal{D}$ is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.   □

In other words, $\mathcal{D}$ is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ iff $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B}$ meets all requirements, i.e. consistency and/or coherency, as per condition (3.1), does not entail any negative test cases as per condition (3.3), and the positive test cases $p \in P$ can be added to $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B}$ without violating any of the conditions (3.1) or (3.3).

From a given DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, a solution KB $\mathcal{K}^*$ can be obtained by a deletion and an expansion step. The deletion step involves the elimination of a diagnosis $\mathcal{D} \subseteq \mathcal{K}$ from $\mathcal{K}$. Note that, due to monotonicity of $\mathcal{L}$, only deletion (and not expansion) of the KB can effectuate a repair of inconsistencies, incoherencies and unwanted entailments. Note, if $\mathcal{K}$ is already valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, then $\mathcal{D}$ can be set to $\emptyset$ and the deletion step can be omitted. The expansion step aims at the fulfillment of positive test cases P, i.e. condition (3.2), which is not necessarily the case after the deletion step. In fact, some new logical sentences $\mathcal{E} \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ may need to be added to $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B}$ to grant entailment of all positive test cases.

Corollary 3.1. *Let $\mathcal{D}$ be a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then there is a set of logical sentences $\mathcal{E} \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ over $\mathcal{L}$ such that:*

$$
\begin{aligned}
\forall\, r \in R &\quad : \quad (\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E} \cup \mathcal{B} \text{ fulfills } r \\
\forall\, p \in P &\quad : \quad (\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E} \cup \mathcal{B} \models p \\
\forall\, n \in N &\quad : \quad (\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E} \cup \mathcal{B} \not\models n.
\end{aligned}
$$

Proof. The proposition of the corollary is a direct consequence of Definition 3.2 and Definition 3.5.   □

From the point of view of a solution KB $\mathcal{K}^*$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, $\mathcal{K} \setminus \mathcal{K}^*$ is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $\mathcal{K}^* \setminus \mathcal{K}$ is one possible extension w.r.t. $\mathcal{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Proposition 3.3. *For each solution KB $\mathcal{K}^*$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ there is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and an extension $\mathcal{E}$ w.r.t. $\mathcal{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ such that $\mathcal{K}^* = (\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E}$ and $\mathcal{E} \cap \mathcal{D} = \emptyset$.*

Proof. Let $\mathcal{K}^*$ be a solution KB w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then $\mathcal{K}^*$ can be written as $\mathcal{K}^* = (\mathcal{K} \cap \mathcal{K}^*) \cup (\mathcal{K}^* \setminus \mathcal{K}) = (\mathcal{K} \setminus (\mathcal{K} \setminus \mathcal{K}^*)) \cup (\mathcal{K}^* \setminus \mathcal{K})$. Let $\mathcal{K} \setminus \mathcal{K}^* =: \mathcal{D}$ and $\mathcal{K}^* \setminus \mathcal{K} =: \mathcal{E}$, then $\mathcal{E} \cap \mathcal{D} = \emptyset$. Further on, $\mathcal{D} \subseteq \mathcal{K}$ holds and $\mathcal{E}$ is a set of logical sentences such that $\mathcal{K}^* = (\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E} \in \mathbf{Sol}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. Therefore, $\mathcal{D} \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ and $\mathcal{E} \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$.   □

Corollary 3.2. *The (non-)existence of a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is equivalent to the (non-)existence of a solution KB w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Proof. Proposition 3.3 shows that there is a diagnosis for each solution KB. By Definition 3.5, there is also a solution KB for each diagnosis. $\square$

The next Proposition gives sufficient and necessary criteria for the existence of a solution, i.e. a diagnosis or a solution KB, respectively, for a given DPI.

Proposition 3.4. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI. Then, a diagnosis $\mathcal{D}$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ exists iff*

- $\forall\, r \in R \ : \ \mathcal{B} \cup U_P$ *fulfills r and*

- $\forall\, n \in N \ : \ \mathcal{B} \cup U_P \not\models n.$

Proof. "$\Leftarrow$": Let us define $\mathcal{D} := \mathcal{K}$. Then $X := (\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup U_P = \mathcal{B} \cup U_P$. Consequently, X satisfies each $r \in R$ as per condition (3.1), $X \not\models n$ for each $n \in N$ as per condition (3.3), and finally $X \models p$ for each $p \in P$ by extensiveness of $\mathcal{L}$ and thus meets condition (3.2). So, X is a solution KB w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ wherefore $\mathcal{D}$ must be a diagnosis.

"$\Rightarrow$": Let $\mathcal{D} \subseteq \mathcal{K}$ be some diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then, by definition of a diagnosis, there is some solution KB $\mathcal{K}^*$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then $\mathcal{K}^* \cup \mathcal{B} \models p$ for all $p \in P$ by condition (3.2), which implies that $\mathcal{K}^* \cup \mathcal{B} \cup U_P$ does not feature any new entailments compared to $\mathcal{K}^* \cup \mathcal{B}$ by idempotency of $\mathcal{L}$. So, $\mathcal{K}^* \cup \mathcal{B} \equiv \mathcal{K}^* \cup \mathcal{B} \cup U_P$ holds. Now, for arbitrary $n \in N$, since $\mathcal{K}^* \cup \mathcal{B} \not\models n$ we have that $\mathcal{K}^* \cup \mathcal{B} \cup U_P \not\models n$, and, by monotonicity of $\mathcal{L}$, that $\mathcal{B} \cup U_P \not\models n$. Analogously, for any $r \in R$, because $\mathcal{K}^* \cup \mathcal{B}$ satisfies r, it must be true that $\mathcal{K}^* \cup \mathcal{B} \cup U_P$ satisfies r and, by monotonicity of $\mathcal{L}$, that $\mathcal{B} \cup U_P$ satisfies r. $\square$

Definition 3.6 (Admissible DPI). *We call a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ admissible iff there is at least one diagnosis $\mathcal{D} \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}.$*

A non-admissible DPI may arise in a situation where a user specifies test cases manually. For this procedure a similar error-proneness as for the user's formulation of KB formulas can be assumed. And there are lots of pitfalls to escape, as Proposition 3.4 shows. In particular, the specified test cases in P and N must be "compatible" with each other, i.e. positive test cases must not contradict negative ones. For example, adding $p_1 := \{A \sqsubseteq C, E \equiv B\}$ and $p_2 := \{C \sqsubseteq E\}$ to P and $n_1 := \{A \sqsubseteq B\}$ to N leads to a contradiction between P and N and consequently to the non-admissibility of a DPI comprising P and N. Furthermore, the background KB $\mathcal{B}$ which is considered as correct, must indeed be correct, at least in terms of R; and negative test cases must be specified in a way not to postulate non-entailment of knowledge specified in $\mathcal{B}$. A counterexample is $\mathcal{B} := \{\exists r.\top \sqsubseteq A, r(x,y), A \sqsubseteq C\}$ and $N := \{\{C(x)\}\}$. And third, the union of positive test cases together with $\mathcal{B}$ must be in compliance with R, particularly the formulas in P must not be inconsistent or incoherent. Because the union of positive test cases U_P can be viewed as an own KB since all logical sentences occurring in some $p \in P$ must be true in the solution KB. So, in a setting where test cases are specified manually, faults occur as likely in U_P as they do in $\mathcal{K}$.

The debugging system presented in this work, however, guarantees by automatic test case generation that admissibility of a DPI is satisfied at any time, provided that an admissible DPI is given as an initial input to the debugging system.

Remark 3.3 In case of a present DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ which is non-admissible, the DPI must be properly modified before it can be used with our debugging system. More concretely, the sets $\mathcal{B}, P$ as well as N must be prepared in a way that the two conditions in Proposition 3.4 are satisfied. When supposing that $\mathcal{B}$ is an already approved and correct KB (which is a reasonable assumption for a KB used as background knowledge during a debugging session), then there are (at least) the following ways to obtain an admissible DPI from a given non-admissible DPI without modifying $\mathcal{B}$.

(a) One straightforward way to achieve that is the deletion of all manually specified test cases from P and N. After that, both sets are either the empty set (if no automatic test cases, e.g. from former

debugging sessions were included in these sets) or comprise only automatically generated test cases. The former case yields an admissible DPI independently of $\mathcal{K}$ by the property of $\mathcal{B}$ to not violate any requirements in R (see Definition 3.1). That the latter case implies the admissibility of the DPI is a property of the debugging system described in this work (as we will show later by Corollary 7.3).

(b) Another way to resolve the non-admissibility of a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is to first check whether $\langle U_P, \mathcal{B}, \emptyset, N \rangle_R$ is admissible (verification of Proposition 3.4 by means of a reasoning service). If so, it is clear that $\mathcal{B}$ does not conflict with N. Then, a debugger (like the one presented in this work) can be exploited to find an as small as possible subset of the set of all formulas occurring in the positive test cases, the removal of which causes the DPI to become admissible. This would be accomplished by the computation of a minimal diagnosis $\mathcal{D}_P$ w.r.t. $\langle U_P, \mathcal{B}, \emptyset, N \rangle_R$ and the usage of the modified admissible DPI $\langle \mathcal{K}, \mathcal{B}, \{U_P \setminus \mathcal{D}_P\}, N \rangle_R$ instead of the original one. In this case, only a set-minimal set $\mathcal{D}_P$ of formulas that were desired entailments of the user are lost. This modification is possible in polynomial time apart from the reasoning costs, i.e. by means of a polynomial number of calls to a reasoner (cf. Chapter 1).

(c) Otherwise, i.e. if $\mathcal{B}$ already conflicts with the negative test cases N, then an algorithm similar to Algorithm 1 (that will be presented in Section 4.4.1) can be employed to determine a maximal subset N' of N w.r.t. set inclusion such that $\mathcal{B}$ will not be in conflict with N'. This approach also requires only a polynomial number of calls to a reasoner (cf. Proposition 4.8). If the resulting modified DPI $\langle \mathcal{K}, \mathcal{B}, P, N' \rangle_R$ is not yet admissible, i.e. after adding the positive test cases U_P to $\mathcal{B}$ there *are* again conflicts with N', method (b) must be executed in order to finally obtain an admissible DPI.

That is, given a non-admissible DPI, there is a transformation achievable in polynomial time which enables the establishment of admissibility involving a set-minimal number of modifications to the given test cases. Thence, in the rest of this work, we will assume that a DPI given as an input to our algorithms is admissible. $\square$

In general, there are multiple (minimal) diagnoses for a DPI, i.e. $|\mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}| \geq |\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}| > 1$, and there are multiple, in fact infinitely many, extensions $\mathcal{E} \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ for a fixed diagnosis $\mathcal{D} \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. The task addressed in this work is finding an optimal diagnosis for a given DPI, whereas the identification of an *optimal* extension w.r.t. that diagnosis and the DPI is not the aim. What we understand by "optimality" of a diagnosis will be addressed in more detail in Part II. Instead, we will content ourselves with finding any extension that enables to formulate a solution KB given a DPI and a diagnosis for that DPI. In fact, the problem of finding a solution KB for a DPI can be reduced to finding a diagnosis for that DPI since a suitable extension can be easily formulated for any diagnosis, as the next proposition shows:

Proposition 3.5. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI and $\mathcal{D} \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. Then U_P is an extension w.r.t. $\mathcal{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Proof. Let us assume that there is some $\mathcal{D} \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ and U_P is not an extension w.r.t. $\mathcal{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. By the definition of a diagnosis, this is equivalent to stating that $(\mathcal{K} \setminus \mathcal{D}) \cup U_P$ is not a solution KB which in turn means that at least one condition (3.1), (3.2) or (3.3) of Definition 3.2 is violated by $(\mathcal{K} \setminus \mathcal{D}) \cup U_P$. However, the fact that $\mathcal{D}$ is a diagnosis implies the existence of some extension $\mathcal{E} \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ that can be added to $(\mathcal{K} \setminus \mathcal{D})$ to obtain a solution KB. This means that conditions (3.1) and (3.3) must be already valid for $(\mathcal{K} \setminus \mathcal{D})$, since, by monotonicity of $\mathcal{L}$, addition of logical sentences $\mathcal{E}$ can neither solve inconsistencies or incoherencies necessary for fulfillment of condition (3.1) nor invalidate non-desired entailments as per condition (3.3). As a consequence, condition (3.2) must be violated by $(\mathcal{K} \setminus \mathcal{D}) \cup U_P$. By extensiveness of $\mathcal{L}$ it holds that $(\mathcal{K} \setminus \mathcal{D}) \cup U_P \models p$ for all $p \in P$ whereby we obtain that condition (3.2) is fulfilled which yields a contradiction. $\square$

Proposition 3.5 claims that the expansion operation, i.e. identifying a concrete extension for a diagnosis, is trivial, at least for our purposes, namely formulating an extension reflecting only evident

entailments given by the set of positive test cases P. Consequently, in order to find a solution KB for some DPI, it is sufficient to concentrate on the deletion step, i.e. on the search for diagnoses.

Note that using U_P as a canonical extension when computing diagnoses does not affect the set of identified diagnoses. In other words, exchanging $\mathcal{E} \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ for U_P in Definition 3.5 yields an equivalent definition. The following corollary proves this statement and summarizes the relationship between the notions *diagnosis*, *solution KB* and *valid KB*.

Corollary 3.3. *The following statements are equivalent:*

1. $\mathcal{D}$ *is a diagnosis w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$

2. $(\mathcal{K} \setminus \mathcal{D}) \cup U_P$ *is a solution KB w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$

3. $(\mathcal{K} \setminus \mathcal{D})$ *is valid w.r.t.* $\langle \cdot, \mathcal{B}, P, N \rangle_R$.

Proof. That (1) is equivalent to (2) follows from Definition 3.5 which states that $\mathcal{D}$ is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ iff there is some set of sentences $\mathcal{E} \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ such that $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E}$ is a solution KB, and from Proposition 3.5 which proves that U_P is an extension w.r.t. *any* diagnosis $\mathcal{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

That (1) is equivalent to (3) follows directly from Proposition 3.2 and the equivalence of (2) and (3) has been shown in Proposition 3.1. □

3.1 Parsimonious Knowledge Base Debugging

Why are *minimal* diagnoses interesting? First, the set of minimal diagnoses w.r.t. a DPI captures all the information that explains the unwanted properties, i.e. violation of requirements or test cases, of the DPI. In other words, the minimal diagnoses represent all subset-minimal possibilities to modify a KB in a way it becomes a valid KB w.r.t. the given DPI (e.g. by simply deleting a minimal diagnosis from the KB in the trivial case). By monotonicity of the logic $\mathcal{L}$, each superset of a minimal diagnosis w.r.t. a DPI is a diagnosis w.r.t. this DPI. That is, $\mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ can be easily reconstructed given $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. There is however no evidence (in terms of specified requirements and test cases) in a DPI that would justify the selection of a non-minimal diagnosis. That is, if $\mathcal{K}$ is a KB and $\mathcal{D} \subseteq \mathcal{K}$ a minimal diagnosis w.r.t. a DPI including $\mathcal{K}$, $\mathcal{K} \setminus \mathcal{D}$ does not violate any of the postulated properties that must hold for a KB to be valid w.r.t. this DPI. For that reason, there is no *evident* need to delete or modify any other sentences in $\mathcal{K}$ except for the ones in some *minimal* diagnosis $\mathcal{D}$.

Second, usually a setting can be assumed where the author of a KB specifies formulas to the best of their knowledge. Hence, the assumption that a formula is rather correct than faulty, or in other words, that the KB author wants to keep as many formulated sentences as possible in a solution KB obtained from a debugger, is practical.

This also motivates the importance of a certain subset of minimal diagnoses, namely minimum cardinality diagnoses, which are the solutions of choice in scenarios where no probabilistic information about the KB authors' faults is available, e.g. in terms of statistics retrieved from log data of the used IDE (see Section 4.6 for details). In an application where such information is given, minimum cardinality diagnoses might not always be the appropriate choice (for details see Part II). In this case the aim is to find a minimal diagnosis with a maximal probability of including only sentences that are actually faulty (which might not necessarily be a minimum cardinality diagnosis).

Third, minimality of diagnoses will be a necessary condition to *guarantee* the possibility of discrimination between different (candidate) diagnoses to formulate a solution KB, as will be seen later in Chapter 7.

Fourth, focusing only on minimal diagnoses rather than all diagnoses can greatly reduce the search space for diagnoses and therefore greatly speed up the debugging procedure (cf. [dKW87]).

Projected to the task of KB debugging, namely finding a solution KB w.r.t. a given DPI, this means we are interested in minimal invasiveness, that is making as few formula-deletion-modifications to the input KB $\mathcal{K}$ as possible in the course of the performed debugging actions. That is, the actual goal is to find some *maximal* solution KB $\mathcal{K}^*$ for a DPI. Compare with "The Principle of Parsimony" in [Rei87, p. 7] [BATJ91].

Problem Definition 3.2 (Parsimonious KB Debugging). *Given a DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, *the task is to find a maximal solution KB w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

The next proposition shows that this problem can be reduced to finding a minimal diagnosis.

Proposition 3.6. *(i)* $\mathcal{K} \setminus \mathcal{K}^*$ *is a minimal diagnosis w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *for each maximal solution KB* $\mathcal{K}^*$ *w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

(ii) If $\mathcal{D}$ *is a minimal diagnosis w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, *then* $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E}$ *is a maximal solution KB w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *for all extensions* $\mathcal{E} \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$.

Proof. **Ad (i):** Let $\mathcal{K}^*$ be an arbitrary maximal solution KB w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. The first observation is that $\mathcal{D} := \mathcal{K} \setminus \mathcal{K}^*$ is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ since $\mathcal{K}^* \setminus \mathcal{K} \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ by the fact that $\mathcal{K}^* = (\mathcal{K} \setminus \mathcal{D}) \cup (\mathcal{K}^* \setminus \mathcal{K})$ is a solution KB by assumption. Let us assume that there is a diagnosis $\mathcal{D}_k \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ such that $\mathcal{D} \supset \mathcal{D}_k$. Since $\mathcal{D}_k$ is a diagnosis, it holds per Definition 3.5 that there is an extension $\mathcal{E} \in \mathbf{EX}(\mathcal{D}_k)_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ such that $\mathcal{K}_k^* := (\mathcal{K} \setminus \mathcal{D}_k) \cup \mathcal{E}$ is a solution KB. Further on, $\mathcal{K} \cap \mathcal{K}_k^* = \mathcal{K} \cap ((\mathcal{K} \setminus \mathcal{D}_k) \cup \mathcal{E}) = (\mathcal{K} \setminus \mathcal{D}_k) \cup (\mathcal{K} \cap \mathcal{E})$. Since $\mathcal{K} \cap \mathcal{K}^*$ can be written as $\mathcal{K} \setminus (\mathcal{K} \setminus \mathcal{K}^*) = \mathcal{K} \setminus \mathcal{D}$ which is a strict subset of $\mathcal{K} \setminus \mathcal{D}_k$ which in turn is a subset of $(\mathcal{K} \setminus \mathcal{D}_k) \cup (\mathcal{K} \cap \mathcal{E}) = \mathcal{K} \cap \mathcal{K}_k^*$. Consequently, $\mathcal{K} \cap \mathcal{K}^* \subset \mathcal{K} \cap \mathcal{K}_k^*$ holds, which is by Definition 3.2 a contradiction to the maximality of the solution KB $\mathcal{K}^*$. Thus, $\mathcal{D} = \mathcal{K} \setminus \mathcal{K}^*$ is a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Ad (ii): Let $\mathcal{D}$ be a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then, by Definition 3.5, there is an extension $\mathcal{E} \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ such that $\mathcal{K}^* := (\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E}$ is a solution KB. Let us assume that $\mathcal{E} \cap \mathcal{D} \neq \emptyset$. We can rewrite $\mathcal{K}^*$ as $\mathcal{K}^* = (\mathcal{K} \setminus \mathcal{D}) \cup (\mathcal{E} \cap \mathcal{D}) \cup (\mathcal{E} \setminus \mathcal{D})$. Since $\emptyset \subset \mathcal{E} \cap \mathcal{D} \subseteq \mathcal{D}$, we have that $(\mathcal{K} \setminus \mathcal{D}) \cup (\mathcal{E} \cap \mathcal{D}) \supset \mathcal{K} \setminus \mathcal{D}$. Thus, there is a $\mathcal{D}' := \mathcal{D} \setminus (\mathcal{E} \cap \mathcal{D}) \subset \mathcal{D}$ and an extension $\mathcal{E}' \in \mathbf{EX}(\mathcal{D}')_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ such that $\mathcal{E}' := \mathcal{E} \setminus \mathcal{D}$ such that $\mathcal{K}^* = (\mathcal{K} \setminus \mathcal{D}') \cup \mathcal{E}'$. As $\mathcal{K}^*$ is a solution KB, this is a contradiction to the minimality of $\mathcal{D}$. Therefore, (*) $\mathcal{E} \cap \mathcal{D} = \emptyset$ for all $\mathcal{E} \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ must hold.

Let $\mathcal{E}$ be any extension w.r.t. $\mathcal{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then we can write $\mathcal{K} \cap \mathcal{K}^* = \mathcal{K} \cap ((\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E}) = (\mathcal{K} \setminus \mathcal{D}) \cup (\mathcal{K} \cap \mathcal{E})$ and by (*) also $\mathcal{K} \cap \mathcal{E} = ((\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{D}) \cap \mathcal{E} = ((\mathcal{K} \setminus \mathcal{D}) \cap \mathcal{E}) \cup (\mathcal{D} \cap \mathcal{E}) = (\mathcal{K} \setminus \mathcal{D}) \cap \mathcal{E} \subseteq \mathcal{K} \setminus \mathcal{D}$. Consequently, (**) $\mathcal{K} \cap \mathcal{K}^* = \mathcal{K} \setminus \mathcal{D}$. Now, assume that there is a solution KB $\mathcal{K}_k^*$ with the property $\mathcal{K} \cap \mathcal{K}_k^* \supset \mathcal{K} \cap \mathcal{K}^*$. By (**), this implies that $\mathcal{K} \cap \mathcal{K}_k^* \supset \mathcal{K} \setminus \mathcal{D}$ which means that there is a $\mathcal{D}_k \subset \mathcal{D} \subseteq \mathcal{K}$ such that $\mathcal{K} \cap \mathcal{K}_k^* = \mathcal{K} \setminus \mathcal{D}_k \subseteq \mathcal{K}_k^*$. Now $\mathcal{K}_k^*$ is a solution KB w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and can be written as $\mathcal{K}_k^* = (\mathcal{K}_k^* \cap \mathcal{K}) \cup (\mathcal{K}_k^* \setminus \mathcal{K}) = (\mathcal{K} \setminus \mathcal{D}_k) \cup (\mathcal{K}_k^* \setminus \mathcal{K})$. By $\mathcal{D}_k \subseteq \mathcal{K}$ and since there is a set of formulas $\mathcal{E} := \mathcal{K}_k^* \setminus \mathcal{K}$ such that $(\mathcal{K} \setminus \mathcal{D}_k) \cup \mathcal{E} \in \mathbf{Sol}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ we have that $\mathcal{E} \in \mathbf{EX}(\mathcal{D}_k)_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ must hold wherefore $\mathcal{D}_k$ is a diagnosis by Definition 3.5. This, however, is a contradiction to the minimality of $\mathcal{D}$. Therefore, $\mathcal{K}^* = (\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{E}$ must be a maximal solution KB for any $\mathcal{E} \in \mathbf{EX}(\mathcal{D})_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. $\square$

By claim (i), Proposition 3.6 assures that each maximal solution KB can be found by investigating all minimal diagnoses w.r.t. a DPI. Claim (ii) shows that any solution KB built from a minimal diagnosis is indeed maximal. Thus, finding a suitable minimal diagnosis solves the problem of parsimonious KB debugging completely.

3.2 Background Knowledge

The general debugging setting considered in this work envisions the opportunity for the user to specify some background knowledge $\mathcal{B}$, i.e. a set of formulas that are known (or strongly assumed) to be correct

in advance. Note that, in order for the debugging procedure to work soundly, before some background knowledge is incorporated into the DPI, it is necessary to verify its conformance with the postulated requirements R (cf. Definition 3.1).We can distinguish between two basic scenarios how background knowledge can be leveraged: (1) We have an initial KB $\mathcal{K}_{init}$ and we know or want to assume that a subset of formulas in $\mathcal{K}_{init}$ is correct, i.e. $\mathcal{B} \cap \mathcal{K}_{init} \neq \emptyset$, and (2) we have an initial KB $\mathcal{K}_{init}$ and some background knowledge disjoint from $\mathcal{K}_{init}$, i.e. $\mathcal{B} \cap \mathcal{K}_{init} = \emptyset$.

Example use cases for scenario (1) are situations where a user knows that a subset of formulas $\mathcal{B}$ in $\mathcal{K}$ is definitely sound or wants to restrict the scope of debugging to a particular part of the KB. Concretely, this may occur, for instance, when $\mathcal{B}$ is the result, i.e. the finally output solution KB $\mathcal{K}^*$, of a former successful debugging session and $\mathcal{K}$ is a further development of $\mathcal{K}^*$, or in a collaborative setting where many users are involved in the development of $\mathcal{K}$ and one of them may want to debug only formulas authored by herself and not touch foreign formulas, which are thus assumed as correct and assigned to $\mathcal{B}$. In (1), $\mathcal{K}_{init} \cap \mathcal{B}$ and $\mathcal{K}_{init} \setminus \mathcal{B}$ partition the original KB $\mathcal{K}_{init}$ into a set of correct and a set of possibly incorrect formulas, respectively. The corresponding DPI would thus be $\langle \mathcal{K}_{init} \setminus \mathcal{B}, \mathcal{B}, P, N \rangle_R$ for some sets of test cases P and N. Note that this DPI *does* meet the necessary condition (cf. Definition 3.1) $\mathcal{K} \cap \mathcal{B} = \emptyset$ as $(\mathcal{K}_{init} \setminus \mathcal{B}) \cap \mathcal{B} = \emptyset$. So, in the debugging session, only $\mathcal{K} := \mathcal{K}_{init} \setminus \mathcal{B}$ is used to search for diagnoses, which can reduce the search space substantially. Though, $\mathcal{B}$ is incorporated in the calculations throughout the KB debugging procedure, but no formula in $\mathcal{B}$ may take part in a diagnosis. The advantage of this over simply not considering the formulas in $\mathcal{B}$ at all is, that the semantics of formulas in $\mathcal{B}$ is not lost and can be exploited, e.g., to grant the desired semantic properties also in the context of existing approved knowledge or to facilitate a greater choice of queries to interact with a user, which can be exploited to ask queries with lower cardinality or involving less complex formulas (see Chapter 7 for details on queries).

In scenario (2), the corresponding DPI looks like $\langle \mathcal{K}_{init}, \mathcal{B}, P, N \rangle_R$ for some sets of test cases P and N. An application of this scenario could be the reuse of an existing KB to support an increase of the fault detection rate and thus more sustainable debugging. For example, when formulating a KB $\mathcal{K}_{init}$ about a domain, a reference KB $\mathcal{B}$ in that domain that is thoroughly curated by experts could be leveraged. The use of such a KB $\mathcal{B}$ is possible both if $\mathcal{K}_{init}$ is correct as a standalone KB, i.e. $\mathcal{K}_{init}$ is already a solution KB for $\langle \mathcal{K}_{init}, \emptyset, P, N \rangle_R$, or not. In the first case, $\mathcal{K}_{init}$ might still contain formulations conflicting with $\mathcal{B}$. In this vein, in both cases, faults may be detected that would have been missed otherwise.

Chapter 4

Diagnosis Computation

In this chapter we describe methods for computing minimal diagnoses w.r.t. a given admissible DPI, provide an in-depth theoretical analysis of these methods including correctness proofs and illustrate the presented algorithms by various examples.

4.1 Conflict Sets

The search space for minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ the size of which is in general $O(2^{|\mathcal{K}|})$ (if all subsets of the KB $\mathcal{K}$ are investigated) can be reduced to a great extent by exploiting the notion of a conflict set [Rei87, dKW87, SFFR12].

Definition 4.1 (Conflict Set). *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI. A set of formulas $\mathcal{C} \subseteq \mathcal{K}$ is called a* conflict set *w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, written as $\mathcal{C} \in \mathbf{aC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, iff $\mathcal{C} \cup U_P$ is not a solution KB w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. A conflict set $\mathcal{C}$ is minimal, written as $\mathcal{C} \in \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, iff there is no $\mathcal{C}' \subset \mathcal{C}$ such that $\mathcal{C}'$ is a conflict set.*

Simply put, a (minimal) conflict set is a (minimal) faulty KB that is a subset of $\mathcal{K}$. That is, a conflict set is one source causing the faultiness of $\mathcal{K}$ in the context of $\mathcal{B} \cup U_P$. In other words, a valid KB may not include all the formulas of any conflict set.

Corollary 4.1. *$\mathcal{C} \subseteq \mathcal{K}$ is a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ iff $\mathcal{C}$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$.*

Proof. If $\mathcal{C}$ is a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, then $\mathcal{C} \cup U_P$ is not a solution KB, i.e. $\mathcal{C} \cup \mathcal{B} \cup U_P$ violates some $r \in R$, some $p \in P$ or some $n \in N$. By extensiveness of $\mathcal{L}$, $\mathcal{C} \cup \mathcal{B} \cup U_P \models p$ for all $p \in P$, so $\mathcal{C} \cup \mathcal{B} \cup U_P$ must violate some $r \in R$ or entail some $n \in N$. Thus, by Definition 3.3, $\mathcal{C}$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$.

If $\mathcal{C} \subseteq \mathcal{K}$ is not valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, then $\mathcal{C} \cup \mathcal{B} \cup U_P$ violates some $r \in R$ or entails some $n \in N$, wherefore $\mathcal{C} \cup U_P \notin \mathbf{Sol}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. Hence, by Definition 4.1, $\mathcal{C}$ is a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. $\square$

Consequently, a conflict set $\mathcal{C}$ along with the background knowledge $\mathcal{B}$ either violates some $r \in R$, entails some $n \in N$, or yields to a violation of some $r \in R$ or entailment of some $n \in N$ if all formulas U_P comprised by the positive test cases are added to $\mathcal{C}$. Any KB $\mathcal{K}$ that is not valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$ is itself a conflict set and includes at least one minimal conflict set.

Proposition 4.1. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI. Then, $\mathcal{K}$ is not valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$ iff $\mathcal{K}$ includes at least one minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Proof. "$\Rightarrow$": Let $\mathcal{K}$ be not valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. Then $\mathcal{K} \cup U_P$ is not a solution KB w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, which means that $\mathcal{K}$ is a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ by definition 4.1. So, either $\mathcal{K}$ is a already a minimal conflict set or there must be some subset $\mathcal{C} \subset \mathcal{K}$ which is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

"$\Leftarrow$": Let $\mathcal{K}$ include at least one minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then, by Definition 4.1, there is some $\mathcal{C} \subseteq \mathcal{K}$ such that $\mathcal{C} \cup U_P$ is not a solution KB. Hence, by the monotonicity of $\mathcal{L}$, $\mathcal{K} \cup U_P$ cannot be a solution KB either. So, by Proposition 3.1, $\mathcal{K}$ is not valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. $\qquad\square$

As a consequence, a complete and sound method for computing minimal conflict sets w.r.t. a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ can be used to decide validity of $\mathcal{K}$ w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. Moreover, such a method can be used to decide whether a given DPI is admissible, i.e. has solutions. For, if a DPI is admissible and the given KB is invalid w.r.t. this DPI, then there cannot be an empty conflict set. In other words, if the empty KB is a conflict set – or, equivalently, an empty conflict set exists w.r.t. a DPI –, then the DPI is not admissible.

Proposition 4.2. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI and $\mathcal{K}$ be invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. Then, there exists a minimal conflict set $\mathcal{C} \neq \emptyset$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ iff $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is admissible.*

Proof. Since $\mathcal{K}$ is not valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, there must be at least one conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ by Proposition 4.1. Assume that there exists a minimal conflict set $\mathcal{C} \neq \emptyset$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. This can be true iff $\emptyset$ is not a (minimal) conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. By Corollary 4.1 and Definition 3.3, this is equivalent to the fact that $\emptyset \cup \mathcal{B} \cup U_P \equiv \mathcal{B} \cup U_P$ does not violate any $r \in R$ and does not entail any $n \in N$. By Proposition 3.4, this holds iff there exists a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. By Definition 3.6, this is equivalent to $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ being admissible. $\qquad\square$

The following proposition provides information about the relationship between (minimal) conflict sets and the background knowledge as well as the positive test cases.

Proposition 4.3. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI and $\mathcal{C}$ a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then the following holds:*

1. *$\mathcal{C} \cap \mathcal{B} = \emptyset$.*

2. *If $\mathcal{C}$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, then $\mathcal{C} \cap U_P = \emptyset$.*

Proof. 1): $\mathcal{C} \cap \mathcal{B} = \emptyset$ holds since $\mathcal{C} \subseteq \mathcal{K}$ (Definition 4.1) and $\mathcal{K} \cap \mathcal{B} = \emptyset$ (Definition 3.1).

2): Assume that $\mathcal{C}$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $\mathcal{C} \cap U_P \neq \emptyset$. Since $\mathcal{C}$ is a conflict set, we have that $\mathcal{C} \cup \mathcal{B} \cup U_P$ violates some $r \in R$ or entails some $n \in N$ by Corollary 4.1 and Definition 3.3. Since $(\mathcal{C} \setminus U_P) \cup \mathcal{B} \cup U_P = \mathcal{C} \cup \mathcal{B} \cup U_P$ and $(\mathcal{C} \setminus U_P) \subset \mathcal{C}$, this implies that $(\mathcal{C} \setminus U_P)$ is a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ which in turn implies that $\mathcal{C} \notin \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ which is a contradiction. $\qquad\square$

4.2 Conflict Sets versus Justifications

The notion of a conflict set is closely related to the notion of a justification [HPS08, HPS09, HPS10, Hor11, HBP11, HPS12a] which is frequently adopted in the field of the Semantic Web (cf. Section 2.2) in order to find minimal explanations for particular entailments in DL ontologies. Thus, the paradigm of a justification can be a useful aid in the debugging of faulty ontologies [Kal06]. Note that sometimes justifications are referred to as MinAs (Minimal Axiom Sets) [BP08] or MUPS (Minimal Unsatisfiability Preserving Sub-TBoxes) [SHCH07] where the latter term is mostly used in the context of ontology debugging. The notion of a (minimal) conflict set, on the other hand, has been mainly adopted in the Diagnosis community [Rei87, dKW87, PW03, WSM02, FFJS04]. In this section we want to establish

a relationship between these two widely used instruments used for debugging. It will turn out that both terms are strongly related, but in debugging systems like the ones proposed in our work conflict sets are better suited as they automatically focus *only* on the minimal explanations for *faults* in a KB.

For example, the author of [Kal06] i.a. discusses the use of justifications to aid the debugging of incoherent ontologies, i.e. ontologies that include unsatisfiable concepts (cf. Section 2.2). If there are multiple unsatisfiable concepts, then some of these might be only unsatisfiable due to the unsatisfiability of another concept. Assume, for instance, an incoherent DL KB $\mathcal{K} := \{A \sqsubseteq B, B \sqsubseteq E \sqcap \neg E\}$. In $\mathcal{K}$ there are two unsatisfiable concepts A and B where A's unsatisfiability is dependent on B's unsatisfiability. Using the terminology of [Kal06, Hor11], A would be called a *purely derived* unsatisfiable concept whereas B would be called a *root* unsatisfiable concept. Because the (only) justification for the unsatisfiability of A is $J_A := \mathcal{K}$ whereas the (only) justification for the unsatisfiability of B is $J_B = \{B \sqsubseteq E \sqcap \neg E\} \subset J_A$. Therefore, [Kal06] proposes to resolve root unsatisfiable concepts first since this might resolve some (purely) derived concepts as well, as in this example. However, finding out whether a concept is root or derived involves the computation of justifications for all unsatisfiable concepts in a KB. On the other hand, reliance on minimal conflict sets would implicate a direct focus on the faultiness (in this example: the incoherency) *of the KB* and not necessarily on the exact explanations of all unsatisfiable concepts that cause the incoherency. In this vein, no justification for a purely derived concept can be a minimal conflict set. So, the computation of minimal conflict sets involves only the determination of those justifications for faults that *must necessarily* be resolved. Therefore, for the given example, the only minimal conflict set is J_B.

A justification for a given formula (axiom) relative to a KB is a (subset-)minimal subset of the KB that entails the given formula.

Definition 4.2 (Justification for a Formula). *[KPHS07] Let $\mathcal{K}$ be a KB and α a formula, both over $\mathcal{L}$. Then $J \subseteq \mathcal{K}$ is called a justification for α w.r.t. $\mathcal{K}$, written as $J \in \mathsf{Just}(\alpha, \mathcal{K})$, iff $J \models \alpha$ and for all $J' \subset J$ it holds that $J' \not\models \alpha$.*

Since we consider test cases which are *sets of formulas* over $\mathcal{L}$, we generalize the definition of a justification as follows:

Definition 4.3 (Justification for a Set of Formulas). *Let $\mathcal{K}, \mathcal{K}'$ be KBs over $\mathcal{L}$. Then $J \subseteq \mathcal{K}$ is called a justification for $\mathcal{K}'$ w.r.t. $\mathcal{K}$, written as $J \in \mathsf{Just}(\mathcal{K}', \mathcal{K})$, iff $J \models \mathcal{K}'$ and for all $J' \subset J$ it holds that $J' \not\models \mathcal{K}'$.*[16]

In order to express the connection between justifications and conflict sets, we require yet another generalization of this definition. To this end, the following definition characterizes a justification for a set X of KBs relative to a KB $\mathcal{K}$ as a (subset-)minimal subset of $\mathcal{K}$ such that this subset entails *some* KB in X.

Definition 4.4 (Justification for a Set of Sets of Formulas). *Let $\mathcal{K}$ be a KB over $\mathcal{L}$ and X a set of KBs over $\mathcal{L}$. Then $J \subseteq \mathcal{K}$ is called justification for X w.r.t. $\mathcal{K}$, written as $J \in \mathsf{Just}(X, \mathcal{K})$, iff $J \models \mathcal{K}'$ for some $\mathcal{K}' \in X$ and for all $J' \subset J$ it holds that $J' \not\models \mathcal{K}''$ for all $\mathcal{K}'' \in X$.*

Based on Definition 4.4, the relation between conflict sets and justifications is captured by the following Proposition 4.4. Intuitively, any conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is the part of a justification for a fault that is relevant for the debugging task, where fault refers to an inconsistency (and/or incoherency) and/or a negative test case entailed by $\mathcal{K} \cup \mathcal{B} \cup U_P$. Since debugging focuses on the deletion of KB formulas only, "relevant" in this context refers to the subset of the justification that does not contain any sentences in $\mathcal{B}$ and U_P, but solely sentences from $\mathcal{K}$. Importantly, there may be justifications, in general, the relevant subset of which is not a *minimal* conflict set. The reason why this case can arise in spite of

---

[16]Remember that $J \models \mathcal{K}'$ means that $J \models ax$ for each $ax \in \mathcal{K}'$ (cf. Remark 3.1).

the set-minimality of justifications is that the *relevant* part of a justification (for some set of sentences $\mathcal{K}_1$, e.g. a negative test case $n_1 \in N$) may be a superset of the *relevant* part of another justification (for some other set of sentences $\mathcal{K}_2$, e.g. another negative test case $n_2 \in N$) whereas both justifications are not in a subset-relationship (i.e. contain different sentences from $\mathcal{B}$ and/or U_P). This circumstance is illustrated by the following example:

Example 4.1   Let a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be defined as

$$\mathcal{K} := \{B \sqsubseteq E, E \sqsubseteq \exists r.G\}$$
$$\mathcal{B} := \{A \sqsubseteq B\}$$
$$N := \{\{A \sqsubseteq E\}, \{B \sqsubseteq \exists r.G\}\}$$
$$P := \emptyset$$
$$R := \{\text{consistency}\}$$

We have that $\mathcal{K} \cup \mathcal{B} \cup U_P$ is consistent and thus no requirement in R is violated. But, the two negative test cases are both entailed by $\mathcal{K} \cup \mathcal{B} \cup U_P$ wherefore $\mathcal{K}$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. The set of justifications for the violation of the first negative test case is $J_{n_1} = \{\{A \sqsubseteq B, B \sqsubseteq E\}\}$; for the second one it is $J_{n_2} = \{\{B \sqsubseteq E, E \sqsubseteq \exists r.G\}\}$. The relevant subset of the justification J_1 in J_{n_1} is $J_{1,rel} = \{B \sqsubseteq E\}$ (since $\{A \sqsubseteq B\}$ is in $\mathcal{B}$) whereas the relevant subset of the justification J_2 in J_{n_2} is $J_{2,rel} = \{B \sqsubseteq E, E \sqsubseteq \exists r.G\}$, i.e. $J_{1,rel} \subset J_{2,rel}$ despite that there is no set subset-relationship between J_1 and J_2. Hence, there are two justifications that explain the invalidity of $\mathcal{K}$ w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, but there is only one minimal conflict set $\mathcal{C} = J_{1,rel}$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. $\square$

So, generally, the set of minimal conflict sets w.r.t. a DPI is a subset of the set of justifications for faults in $\mathcal{K} \cup \mathcal{B} \cup U_P$, which is due to the focus on just the parts of justifications that are relevant for the KB debugging task.

Proposition 4.4.   *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI. Additionally, let*

(a) $X := \{\{A_i \sqsubseteq \bot\} \mid A_i \in N_C\} \cup \{\{r_i \sqsubseteq \bot\} \mid r_i \in N_R\} \cup \{\{\top \sqsubseteq \bot\}\} \cup N$
    if $R = \{consistency, coherency\}$ and

(b) $X := \{\{\top \sqsubseteq \bot\}\} \cup N$ *if $R = \{consistency\}$.*[17]

Then the following holds:

1. *If $\mathcal{C}$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, then there is some $J \in \mathsf{Just}(X, \mathcal{K} \cup \mathcal{B} \cup U_P)$ such that $(J \cap \mathcal{K}) \setminus U_P = \mathcal{C}$.*

2. *For all $J \in \mathsf{Just}(X, \mathcal{K} \cup \mathcal{B} \cup U_P)$ it is true that $\mathcal{C} := (J \cap \mathcal{K}) \setminus U_P$ is a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, but not necessarily a minimal one.*

Proof. **1):** Assume that $\mathcal{C} \in \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ and for all $J \in \mathsf{Just}(X, \mathcal{K} \cup \mathcal{B} \cup U_P)$ it holds that $(J \cap \mathcal{K}) \setminus U_P \neq \mathcal{C}$. There are two cases to distinguish between: (a) there is some sentence in $(J \cap \mathcal{K}) \setminus U_P$ that is not in $\mathcal{C}$ and (b) there is some sentence in $\mathcal{C}$ that is not in $(J \cap \mathcal{K}) \setminus U_P$.

Let us first assume (a), i.e. for all $J \in \mathsf{Just}(X, \mathcal{K} \cup \mathcal{B} \cup U_P)$ it holds that there is some sentence ax in $(J \cap \mathcal{K}) \setminus U_P$ that is not in $\mathcal{C}$. Additionally, assume there is a $J \in \mathsf{Just}(X, \mathcal{K} \cup \mathcal{B} \cup U_P)$ such that $J \subseteq \mathcal{C} \cup \mathcal{B} \cup U_P$. We can write J as $J = S_1 \cup S_2 \cup S_3$ for $S_1 := [(J \cap \mathcal{K}) \setminus U_P]$, $S_2 := [J \cap \mathcal{B}]$ and $S_3 := [J \cap U_P]$. Since $J = S_1 \cup S_2 \cup S_3 \subseteq \mathcal{C} \cup \mathcal{B} \cup U_P$ it must hold in particular that $S_1 \subseteq \mathcal{C} \cup \mathcal{B} \cup U_P$ and

---

[17]We use DL notation in this proposition since justifications, as argued, are mostly applied to DL KBs. An equivalent formulation of the proposition for FOL or PL is straightforward (cf. Example 2.1 and Remark 3.2). Note that for PL only (b) is relevant since coherency is not defined for PL. Further, recall that N_C and N_R are defined in Section 2.2.

therefore $ax \in \mathcal{C} \cup \mathcal{B} \cup U_P$. However, $ax \notin \mathcal{C}$ by assumption, $ax \notin \mathcal{B}$ since $ax \in \mathcal{K}$ and $\mathcal{B} \cap \mathcal{K} = \emptyset$, and $ax \notin U_P$ since $ax \in S_1$ and $S_1 \cap U_P = \emptyset$. This is a contradiction. Hence, for all $J \in \mathsf{Just}(X, \mathcal{K} \cup \mathcal{B} \cup U_P)$ it holds that $J \nsubseteq \mathcal{C} \cup \mathcal{B} \cup U_P$. Since X captures all $r \in R$ and $n \in N$, we can conclude that $\mathcal{C}$ is not a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ which is a contradiction to $\mathcal{C} \in \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$.

Let us now assume (b), i.e. for all $J \in \mathsf{Just}(X, \mathcal{K} \cup \mathcal{B} \cup U_P)$ it holds that there is some sentence ax in $\mathcal{C}$ that is not in $(J \cap \mathcal{K}) \setminus U_P$. Since $\mathcal{C}$ is a conflict set and since X captures all $r \in R$ and $n \in N$, we have that $\mathcal{C} \cup \mathcal{B} \cup U_P \models \mathcal{K}'$ for some $\mathcal{K}' \in X$. So, there must be some $J_0 \in \mathsf{Just}(X, \mathcal{K} \cup \mathcal{B} \cup U_P)$ such that $J_0 \subseteq \mathcal{C} \cup \mathcal{B} \cup U_P$. As $\mathcal{C} \in \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, there cannot be any $J \in \mathsf{Just}(X, \mathcal{K} \cup \mathcal{B} \cup U_P)$ with $J \subseteq \mathcal{C}' \cup \mathcal{B} \cup U_P$ for arbitrary $\mathcal{C}' \subset \mathcal{C}$. This must hold in particular for J_0 which implies that $J_0 \cap \mathcal{C} = \mathcal{C}$ which is equivalent to $\mathcal{C} \subseteq J_0$. As (1) $\mathcal{C} \subseteq \mathcal{K}$ (Definition 4.1) and, by Proposition 4.3 and by the fact that $\mathcal{C} \in \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, (2) $\mathcal{C} \cap U_P = \emptyset$, we can conclude that $\mathcal{C} \subseteq (J_0 \cap \mathcal{K}) \setminus U_P$ which is a contradiction since there cannot be a ax in $\mathcal{C}$ that is not in $(J_0 \cap \mathcal{K}) \setminus U_P$.

2): If $J \in \mathsf{Just}(X, \mathcal{K} \cup \mathcal{B} \cup U_P)$, then, by Definition 4.4, $J \models \mathcal{K}'$ for some $\mathcal{K}' \in X$ and $J \subseteq \mathcal{K} \cup \mathcal{B} \cup U_P$. So, $[(J \cap \mathcal{K}) \setminus U_P] \cup \mathcal{B} \cup U_P = (J \cap \mathcal{K}) \cup \mathcal{B} \cup U_P \supseteq J$ wherefore $[(J \cap \mathcal{K}) \setminus U_P] \cup \mathcal{B} \cup U_P \models \mathcal{K}'$ by monotonicity of $\mathcal{L}$. As $\mathcal{K}' \in X$ and X captures all the reasons why some $r \in R$ or some $n \in N$ may not be fulfilled (cf. the discussion in Chapter 3), we have that $[(J \cap \mathcal{K}) \setminus U_P] \cup \mathcal{B} \cup U_P$ violates some $r \in R$ or entails some $n \in N$. This implies that $[(J \cap \mathcal{K}) \setminus U_P] \cup U_P \notin \mathbf{Sol}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. Since $(J \cap \mathcal{K}) \setminus U_P \subseteq \mathcal{K}$ is also true, $(J \cap \mathcal{K}) \setminus U_P \in \mathbf{aC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ by Definition 4.1.

To see that $(J \cap \mathcal{K}) \setminus U_P \notin \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ holds in general, reconsider Example 4.1 where $(J_2 \cap \mathcal{K}) \setminus U_P = J_2 \supset \mathcal{C}$ holds for the justification J_2 and the minimal conflict set $\mathcal{C}$. $\qquad \square$

4.3 The Relation between Conflict Sets and Diagnoses

A minimal conflict set has the property that deletion of any formula in it yields a set of formulas which is correct in the context of $\mathcal{B}$, P, N and R.

Proposition 4.5. *If $\mathcal{C}$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, then $\mathcal{C}'$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$ for each $\mathcal{C}' \subset \mathcal{C}$.*

Proof. Since $\mathcal{C} \in \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, it must hold that $\mathcal{C}' \notin \mathbf{aC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. Then, by Corollary 4.1, $\mathcal{C}'$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. $\qquad \square$

Hence, by deletion of at least one formula from *each* minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, a valid KB can be obtained from $\mathcal{K}$. Thus, a solution KB $(\mathcal{K} \setminus \mathcal{D}) \cup U_P$ can be obtained by calculation of a hitting set $\mathcal{D}$ of all minimal conflict sets in $\mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. The Hitting Set problem is defined as follows:

Definition 4.5 (Hitting Set). *Let $S = \{S_1, \dots, S_n\}$ be a set of sets. Then, H is called a* hitting set *of S iff $H \subseteq U_S$ and $H \cap S_i \neq \emptyset$ for all $i = 1, \dots, n$.*
A hitting set H of S is minimal *iff there is no hitting set H' of S such that $H' \subset H$.*

Proposition 4.6. *[FS05] A (minimal) diagnosis w.r.t. the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is a (minimal) hitting set of all minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Now, we want to contemplate two example DPIs and analyze them regarding the their minimal conflict sets and minimal diagnoses:

Example 4.2 In this example, we analyze the PL DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given by Table 15.3. There are two minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, i.e. $\mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} = \{\mathcal{C}_1, \mathcal{C}_2\} = \{\langle 1, 2, 5 \rangle, \langle 1, 2, 7 \rangle\}$.[18]

---

[18]Please notice that we sometimes write i instead of ax_i for brevity when it is clear what is meant. We will do so in many other examples as well.

Why is $\mathcal{C}_1$ a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$? We recall Definition 4.1 and argue as follows to deduce the entailment $\mathcal{C}_1 \models n_1$ where $n_1 \in N$ (left of the colon: the formulas used in the deduction are underlined; right of the colon: the relevant implications are underlined):

$$
\begin{array}{rl}
\underline{ax_1}: & \underline{A \rightarrow E} \\
\underline{ax_2}: & X \vee \underline{E} \rightarrow F \wedge \underline{Y} \wedge Z \\
\underline{ax_5}: & \underline{Y} \rightarrow \neg A \\
\underline{ax_1, ax_2, ax_5}: & \underline{A \rightarrow \neg A} \equiv \underline{\neg A \vee \neg A} \equiv \underline{\neg A} \\
n_1 \in N: & \underline{\neg A} \qquad \square
\end{array}
$$

Minimality of $\mathcal{C}_2$ is obvious from this argumentation. i.e. we cannot deduce n_1 if any one of the formulas 1, 2 or 5 is omitted, and there is no other fault except for the violation of n_1.

Why is $\mathcal{C}_2$ a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$? We recall Definition 4.1 and argue as follows to deduce the entailment $\mathcal{C}_2 \cup \mathcal{B} \models n_1$ where $n_1 \in N$ (left of the colon: the formulas used in the deduction are underlined; right of the colon: the relevant implications are underlined):

$$
\begin{array}{rl}
\underline{ax_1}: & \underline{A \rightarrow E} \\
\underline{ax_2}: & X \vee \underline{E} \rightarrow F \wedge Y \wedge \underline{Z} \\
\underline{ax_7}: & \underline{Z \rightarrow G} \\
(G \rightarrow \neg A) \in \mathcal{B}: & \underline{G \rightarrow \neg A} \\
\underline{ax_1, ax_2, ax_7}, \mathcal{B}: & \underline{A \rightarrow \neg A} \equiv \underline{\neg A \vee \neg A} \equiv \underline{\neg A} \\
n_1 \in N: & \underline{\neg A} \qquad \square
\end{array}
$$

Minimality of $\mathcal{C}_2$ is obvious from this argumentation. i.e. we cannot deduce n_1 if any one of the formulas 1, 2 or 7 is omitted, and there is no other fault except for the violation of n_1.

There are no further minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. This is fairly easy to see since

- $\mathcal{K} \cup \mathcal{B} \cup U_P = \mathcal{K} \cup \mathcal{B}$ cannot be inconsistent due to the fact that the only negative literal occurring on the righthand side of an implication is $\neg A$ and A does not occur at the righthand side of any implication in $\mathcal{K} \cup \mathcal{B}$,

- there is no other way to deduce n_1 than using a superset of the formulas in $\mathcal{C}_1$ or $\mathcal{C}_2$ and

- n_1 is the only negative test case in N.

Hence, the set of all minimal diagnoses $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\} = \{[1], [2], [5, 7]\}$ is obtained by computing all minimal hitting sets of $\mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} = \{\mathcal{C}_1, \mathcal{C}_2\}$ (cf. Proposition 4.6). $\qquad \square$

Example 4.3 In this example, we analyze the DL DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given by Table 4.2. There are four minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, i.e.

$$
\mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} = \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4\} = \{\langle 1, 2, 5 \rangle, \langle 2, 4, 6 \rangle, \langle 1, 3, 4 \rangle, \langle 1, 5, 6, 8 \rangle\}
$$

Why is $\mathcal{C}_1$ a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$? We recall Definition 4.1 and argue as follows to deduce the entailment $\mathcal{C}_1 \models n_1$ where $n_1 \in N$ (left of the colon: the formulas used in the deduction are underlined;

right of the colon: the relevant implications are underlined):

$$
\begin{aligned}
\underline{ax_1} : &\quad \underline{A \ \sqsubseteq \ B} \\
\underline{ax_2} : &\quad \underline{B \ \sqsubseteq \ G} \\
\underline{ax_5} : &\quad \underline{G \ \sqsubseteq \ K} \\
\underline{ax_1, ax_2, ax_5} : &\quad \underline{A \ \sqsubseteq \ K} \\
\underline{n_1 \in N} : &\quad \underline{A \ \sqsubseteq \ K} \qquad \square
\end{aligned}
$$

Minimality of $\mathcal{C}_1$ is follows from this argumentation. i.e. we cannot deduce n_1 if any one of the formulas 1, 2 or 5 is omitted, and from the fact that we cannot deduce an incoherency (r_2), inconsistency (r_1) or the entailment of any other negative test case $n \in N$ for any KB $\mathcal{C}_1' \cup \mathcal{B} \cup U_P$ for any $\mathcal{C}_1' \subset \mathcal{C}_1$.

Why is $\mathcal{C}_2$ a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$? We recall Definition 4.1 and argue as follows to deduce that $\mathcal{C}_2 \cup \mathcal{B}$ is incoherent and thus violates the requirement $r_2 \in R$ (left of the colon: the formulas used in the deduction are underlined; right of the colon: the relevant implications are underlined):

$$
\begin{aligned}
\underline{ax_2} : &\quad \underline{B \ \sqsubseteq \ G} \\
\underline{ax_6} : &\quad \underline{G \ \sqsubseteq \ \exists r.F} \\
(1) : \ \underline{ax_2, ax_6} : &\quad \underline{B \ \sqsubseteq \ \exists r.F} \\
\underline{ax_4} : &\quad \underline{B \ \sqsubseteq \ \forall r.H} \\
(\underline{H \ \sqsubseteq \ \neg F}) \in \mathcal{B} : &\quad \underline{H \ \sqsubseteq \ \neg F} \\
(2) : \ \underline{ax_4}, \mathcal{B} : &\quad \underline{B \ \sqsubseteq \ \forall r.\neg F} \\
(1) \text{ and } (2) : &\quad \underline{B \ \sqsubseteq \ \bot} \\
r_1 \in R : &\quad \underline{B \ \not\sqsubseteq \ \bot} \qquad \square
\end{aligned}
$$

Since we cannot deduce an incoherency (r_2), inconsistency (r_1) or the entailment of any negative test case $n \in N$ for any KB $\mathcal{C}_2' \cup \mathcal{B} \cup U_P$ for any $\mathcal{C}_2' \subset \mathcal{C}_2$, the minimality of $\mathcal{C}_2$ follows.

Why is $\mathcal{C}_3$ a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$? We recall Definition 4.1 and argue as follows to deduce that $\mathcal{C}_3 \cup \mathcal{B} \cup U_P$ is inconsistent and thus violates the requirement $r_1 \in R$ (left of the colon: the formulas used in the deduction are underlined; right of the colon: the relevant implications are underlined):

$$
\begin{aligned}
\underline{A(x)} \in \mathcal{B} : &\quad \underline{A(x)} \\
\underline{ax_1} : &\quad \underline{A \ \sqsubseteq \ B} \\
(1) : \ \underline{ax_1}, \mathcal{B} : &\quad \underline{B(x)} \\
(2) : \ \underline{p_1 \in P} : &\quad \underline{r(x,y)} \\
\underline{ax_4} : &\quad \underline{B \ \sqsubseteq \ \forall r.H} \\
(3) : \ (1) \text{ and } \underline{ax_4} : &\quad \underline{H(y)} \\
(4) : \ \underline{ax_3} : &\quad \underline{\neg H(y)} \\
(3) \text{ and } (4) : &\quad \nvDash \qquad \square
\end{aligned}
$$

No inconsistency (r_1) or incoherency (r_2) can be derived and no negative test case $n \in N$ is entailed from any $\mathcal{C}_3' \cup \mathcal{B} \cup U_P$ for $\mathcal{C}_3' \subset \mathcal{C}_3$. Hence, $\mathcal{C}_3$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Why is $\mathcal{C}_4$ a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$? We recall Definition 4.1 and argue as follows to deduce the entailment $\mathcal{C}_4 \cup \mathcal{B} \models n_2$ where $n_2 \in N$ (left of the colon: the formulas used in the deduction are

underlined; right of the colon: the relevant implications are underlined):

$$
\begin{aligned}
\underline{ax_8} &: & \underline{L} &\sqsubseteq \underline{G} \\
\underline{ax_6} &: & \underline{G} &\sqsubseteq \underline{\exists r.F} \\
(1) : \underline{ax_6}, \underline{ax_8} &: & \underline{L} &\sqsubseteq \underline{\exists r.F} \\
A(x) \in \mathcal{B} &: & \underline{A(x)} & \\
(2) : \underline{ax_1}, \mathcal{B} &: & \underline{B(x)} & \\
(3) : \underline{ax_5} &: & \underline{G} &\sqsubseteq \underline{K} \\
(1) \text{ and } (2) \text{ and } (3) &: & \underline{L} &\sqsubseteq \underline{\exists r.F},\ \underline{B(x)},\ \underline{G} \sqsubseteq \underline{K} \\
n_1 \in N &: & \underline{L} &\sqsubseteq \underline{\exists r.F},\ \underline{B(x)},\ \underline{G} \sqsubseteq \underline{K} \qquad \square
\end{aligned}
$$

No inconsistency (r_1) or incoherency (r_2) can be derived and no negative test case $n \in N$ is entailed from any $\mathcal{C}_4' \cup \mathcal{B} \cup U_P$ for $\mathcal{C}_4' \subset \mathcal{C}_4$. Thus, $\mathcal{C}_4$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Hence, the set of all minimal diagnoses $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, obtained by computing all minimal hitting sets of $\mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} = \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4\}$ (cf. Proposition 4.6), comprises ten minimal diagnoses $\mathcal{D}_i$ for $i = 1, \ldots, 10$:

$$
\begin{aligned}
\mathcal{D}_1 &= [1, 2] & \mathcal{D}_2 &= [1, 4] \\
\mathcal{D}_3 &= [1, 6] & \mathcal{D}_4 &= [2, 3, 5] \\
\mathcal{D}_5 &= [2, 3, 6] & \mathcal{D}_6 &= [2, 3, 8] \\
\mathcal{D}_7 &= [2, 4, 6] & \mathcal{D}_8 &= [2, 4, 8] \\
\mathcal{D}_9 &= [3, 5, 6] & \mathcal{D}_{10} &= [4, 5]
\end{aligned}
$$

Although the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is very small in size, i.e. number of formulas occurring in it is very small, the reader might agree that it is not trivial on the one hand (1) to realize which subsets of this KB $\mathcal{K}$ are (minimal) conflict sets, (2) to see *that* or *why* a subset of this KB $\mathcal{K}$ along with the background knowledge $\mathcal{B}$ and the union of the positive test cases U_P is a (minimal) conflict set (cf. [HBP11]), and (3) to assess that there are no further minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. This example gives a little bit of an impression that tool assistance in the debugging of KBs is inevitable especially for real-world KBs that are huge in size and/or complex in terms of the expressivity of the used logic or in terms of their "debugging properties", i.e. large number and/or size of minimal conflict sets and/or minimal diagnoses.

A means to handle problems (1) and (3) is provided by some method for the computation of a minimal conflict set (e.g. QX given by Algorithm 1 below, see Section 4.4.1) coupled with a hitting set tree algorithm (e.g. HS described by Algorithm 2 below, see Section 4.5) for the systematic computation of *different* minimal conflict sets, or other mechanisms such as the ALL_JUST_ALG presented in [KPHS07] which computes all justifications for some particular entailment (but, some post-processing of the justifications is necessary to obtain minimal conflict sets, cf. Section 4.2).

Problem (2) and its complexity for humans has been studied in [HBP11] with a focus on justifications in DL or OWL KBs. Since a minimal conflict set can be regarded as the relevant (i.e. potentially faulty) part of a justification for some undesired entailment (i.e. a violated requirement or test case) as we analyzed in Section 4.2, the cognitive complexity model proposed by [HBP11] applies also to minimal conflict sets. Ways to facilitate the understanding of justifications for humans (that might be successfully applied also to conflict sets) have been addressed in [HPS10, HPS09, HPS08]. Moreover, there is an ontology editing browser SWOOP [KPS+06] equipped with a strikeout feature [Kal06] that highlights parts of justifications that are relevant for the entailment by striking out all irrelevant parts. This is more or less the automation of our analyses of the conflict sets by underlining the relevant parts of the formulas in this example and Example 4.2. $\square$

i	ax_i	$\mathcal{K}$	$\mathcal{B}$
1	$A \to E$	$\bullet$	
2	$X \vee E \to F \wedge Y \wedge Z$	$\bullet$	
3	$F \to B$	$\bullet$	
4	$B \to X$	$\bullet$	
5	$Y \to \neg A$	$\bullet$	
6	$B \to Z$	$\bullet$	
7	$Z \to G$	$\bullet$	
8	$G \to \neg A$		$\bullet$
i	$p_i \in P$		
$\times$	$\times$		
i	$n_i \in N$		
1	$\neg A$		
i	$r_i \in R$		
1	consistency		

Table 4.1: Propositional Logic Example DPI

4.4 Methods for Diagnosis Computation

Two common methods employed for the computation of (minimal) diagnoses [SFFR12, RSFF13] are the QuickXPlain algorithm [Jun04] (in short QX) and a hitting set search tree [Rei87, GSW89] (in short HS). Thereby, QX serves as a deterministic method for computing one minimal conflict set w.r.t. a given DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ per call. Since a diagnosis is a hitting set of *all* minimal conflict sets, more than one minimal conflict set is generally required to compute a diagnosis. Due to its determinism, however, QX always computes the same minimal conflict set for the same input DPI. Thus, in order to compute different (or all) minimal conflict sets, the input to QX needs to be varied accordingly. This can be done by means of HS which serves as a search tree to systematically and successively explore all minimal conflict sets w.r.t. an initially given DPI. Note that often not all minimal conflict sets w.r.t. a DPI are necessary to obtain a minimal diagnosis w.r.t. this DPI. This is the case when different minimal conflict sets overlap, i.e. have a non-empty intersection. In the extreme case, when all minimal conflict sets w.r.t. a DPI share some formulas, then the computation of any single minimal conflict set can suffice to obtain a minimal diagnosis, which is actually even a minimum cardinality diagnosis.

Another approach for computing a minimal conflict set (or justification) is the "expand-and-shrink" algorithm presented in [KPHS07]. However, empirical evaluations and a theoretical analysis of the best and worst case complexity of the "expand-and-shrink" method compared to QX performed in [SFJ08] revealed that the latter is preferable over the former.

Also, alternative strategies for the computation of minimal diagnoses have been suggested. One common method is to avoid the indirection of diagnosis computation via minimal conflict sets and use algorithms that determine diagnoses *directly* [SU06], i.e. without the necessity to compute conflict sets. This approach has been applied for the non-interactive debugging of ontologies [DQPS11] and constraints [FSZ11]. In our previous work, we adopted such a direct technique for the interactive debugging of KBs [SFRF14c]. The reason why we stick to the conflict-based approach in this work is that we want to present best-first algorithms that figure out minimal diagnoses in descending order of their probability. This is not (systematically) realizable with a direct approach.

i	ax_i	$\mathcal{K}$	$\mathcal{B}$
1	$A \sqsubseteq B$	•	
2	$B \sqsubseteq G$	•	
3	$\neg H(y)$	•	
4	$B \sqsubseteq \forall r.H$	•	
5	$G \sqsubseteq K$	•	
6	$G \sqsubseteq \exists r.F$	•	
7	$A(x)$		•
8	$L \sqsubseteq G$	•	
9	$H \sqsubseteq \neg F$		•

i	$p_i \in P$		
1	$r(x,y)$		

i	$n_i \in N$		
1	$A \sqsubseteq K$		
2	$L \sqsubseteq \exists r.F, B(x), G \sqsubseteq K$		

i	$r_i \in R$		
1	consistency		
2	coherency		

Table 4.2: Description Logic Example DPI

4.4.1   Computation of a Minimal Conflict Set

The QX algorithm takes a DPI $\langle \mathcal{K}_{\text{orig}}, \mathcal{B}_{\text{orig}}, P, N \rangle_R$ over some monotonic logic $\mathcal{L}$ as input and returns a minimal conflict set $\mathcal{C} \subseteq \mathcal{K}_{\text{orig}}$ w.r.t. $\langle \mathcal{K}_{\text{orig}}, \mathcal{B}_{\text{orig}}, P, N \rangle_R$ as output, if some conflict set exists for the DPI, and 'no conflict' otherwise.

Monotonic Properties.   Basically, QX can be employed to find for an input set X a set-minimal subset $X_{\min} \subseteq X$ that has a certain property *prop* for problems of completely different nature such as propositional unsatisfiability or over-constrainedness of constraint satisfaction problems. The only postulated prerequisite for QX to work correctly is that *prop* is a monotonic property. A property is monotonic if and only if the binary function that returns 1 if the property holds for the input set and 0 otherwise is a monotonic function.

Definition 4.6 (Binary Monotonic Function)**.** *Let X be a set and $f : 2^X \to \{0,1\}$ be a binary function defined for all subsets of X. Then, f is monotonic iff*

$$\forall X', X'' \subseteq X : \ X' \subset X'' \wedge f(X') = 1 \implies f(X'') = 1$$

So, *prop* is monotonic iff, given that *prop* holds for some set X', it follows that *prop* also holds for any superset X'' of X'. Note that, by simple logical transformation, an equivalent statement can be derived from Definition 4.6; namely that, given that *prop* does not hold for some set X'', it follows that *prop* does not hold for any subset X' of X'' either.

As inconsistency and incoherency as well as the entailment of some $n \in N$ over some monotonic language $\mathcal{L}$ are clearly monotonic properties, the following proposition holds.

Proposition 4.7. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI. Then, the invalidity of $\mathcal{K}' \subseteq \mathcal{K}$ w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$ (as per Definition 3.3) is a monotonic property.*

By Corollary 4.1, a (minimal) conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is a (minimal) invalid sub-KB of $\mathcal{K}$ w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. Therefore:

Corollary 4.2. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI. Then, being a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is a monotonic property.*

Thus, QX is applicable for the problem of finding a minimal conflict set w.r.t. a DPI. As we shall see later in Chapter 8, another monotonic property will enable us to apply QX also for the minimization of queries asked to an interacting user in the interactive debugging of KBs.

How QX (Algorithm 1) Works. After verifying that the trivial cases, i.e. $\mathcal{K}_{\text{orig}}$ is already a valid KB w.r.t. $\langle \cdot, \mathcal{B}_{\text{orig}}, P, N \rangle_R$ or $\mathcal{K}_{\text{orig}} = \emptyset$, are not met, a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}_{\text{orig}}, \mathcal{B}_{\text{orig}}, P, N \rangle_R$ must exist. So, the algorithm enters the recursive procedure $\text{QX}'(\emptyset, \langle \mathcal{K}_{\text{orig}}, \mathcal{B}_{\text{orig}}, P, N \rangle_R)$. Note that the parameters P, N, R of QX' are used for validity tests (ISKBVALID, line 9) only and are maintained invariant during the entire recursive execution. In case $\mathcal{K}_{\text{orig}}$ is not a singleton, i.e. it does not hold for sure that $\mathcal{K}_{\text{orig}}$ is an element of a minimal conflict set w.r.t. $\langle \mathcal{K}_{\text{orig}}, \mathcal{B}_{\text{orig}}, P, N \rangle_R$, the idea is to apply a divide-and-conquer strategy to reduce $\mathcal{K}_{\text{orig}}$ into two subproblems and solve one subproblem first, i.e. find a minimal conflict set for this subproblem, and then the second subproblem. The union of the minimal conflict sets found for the subproblems is then a minimal conflict set for the original problem. This division into smaller problems is recursively executed for each subproblem until the trivial case, i.e. the KB of the subproblem that is analyzed includes only one element, occurs. Then this element is an element of a minimal conflict set w.r.t. the original problem.

Simply put, one can imagine that QX takes $\mathcal{K}_{\text{orig}}$, partitions it into $\mathcal{K}_1$ and $\mathcal{K}_2$ and first considers the DPI with KB $\mathcal{K}_2$ and background knowledge $\mathcal{B} \cup \mathcal{K}_1$ (line 16). If the latter already includes a conflict set (second condition in line 9), then $\mathcal{K}_2$ can be safely discarded and does not need to be further considered. Instead, $\mathcal{K}_1$ is further investigated, i.e. the DPI with KB $\mathcal{K}_{1,2}$ and background knowledge $\mathcal{B} \cup \mathcal{K}_{1,1}$ where $\mathcal{K}_{1,1}$ and $\mathcal{K}_{2,2}$ partition $\mathcal{K}_1$. Notice that, in this way, $|\mathcal{K}_2|$ sentences can be dismissed by a single call to ISKBVALID which is the only function in Algorithm 1 that calls a reasoner.

If, on the other hand, $\mathcal{B} \cup \mathcal{K}_1$ includes no conflict set, $\mathcal{K}_2$ is partitioned into $\mathcal{K}_{2,1}$ and $\mathcal{K}_{2,2}$ and the two DPIs, the first with KB $\mathcal{K}_{2,2}$ and background knowledge $\mathcal{B} \cup \mathcal{K}_1 \cup \mathcal{K}_{2,1}$ and the second with KB $\mathcal{K}_{2,1}$ and background knowledge $\mathcal{B} \cup \mathcal{K}_1 \cup \mathcal{C}_{2,2}$, are recursively analyzed where $\mathcal{C}_{2,2}$ is the result computed for the first DPI.

This recursion is executed until encountering a trivial case, i.e. a leaf node of the recursion tree, along each path. Then, the recursion unwinds by building the union of all leaf nodes, i.e. the union of all returned sets for subproblems where a trivial case occurred.

The next example illustrates one execution of QX which computes one minimal conflict set:

Example 4.4 Let us consider the DL example DPI depicted by Table 4.3. We will now demonstrate how a minimal conflict set is computed by Algorithm 1 (see Fig. 4.1). Since $\mathcal{K}$ is not the empty set and not a valid KB w.r.t. the DPI (conditions in lines 4 and 2 are false), $\text{QX}'(\emptyset, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ is called in line 7. This call is illustrated by the root node (node ①) of the recursion tree given in Fig. 4.1 (whereas the evaluations made by QX prior to this call are not depicted in the figure). Notice that each node in the tree shows only the values of $\mathcal{C}, \mathcal{K}$ and $\mathcal{B}$ since all other parameters P, N and R are invariant throughout the entire execution of Algorithm 1.

Due to the fact that $\mathcal{C} = \emptyset$ and $\mathcal{K}$ includes five formulas and is thus not a singleton, $\mathcal{K} = \{ax_1, \ldots, ax_5\}$ is partitioned into $\mathcal{K}_1 = \{ax_1, ax_2, ax_3\}$ and $\mathcal{K}_2 = \{ax_4, ax_5\}$ and QX' is recursively called in line 16 with parameters $\mathcal{C} = \mathcal{K}_1$, $\mathcal{K} = \mathcal{K}_2$ and $\mathcal{B} = \mathcal{B} \cup \{ax_1, ax_2, ax_3\}$ which is expressed in the figure by a left branch to node ②. This call, however, returns $\emptyset$ directly since $\mathcal{B} \cup \{ax_1, ax_2, ax_3\}$ is already invalid w.r.t. $\langle \cdot, \emptyset, P, N \rangle_R$ because $\mathcal{B} \cup \{ax_1, ax_2, ax_3\} \cup U_P = \left\{ A(w), \underline{A(v)}, \underline{s(v,w)} \right\} \cup \{A \sqsubseteq B, B \sqsubseteq E, \underline{B \sqsubseteq} D \sqcap \underline{\neg \exists s.C}\} \cup \{\{B(w)\}\} \models \{\neg C(w)\}$ which is a negative test case, i.e. must not

---

Algorithm 1 QX: Computation of a Minimal Conflict Set

Input: a DPI $\langle \mathcal{K}_{\text{orig}}, \mathcal{B}_{\text{orig}}, P, N \rangle_R$
Output: a minimal conflict set w.r.t. $\langle \mathcal{K}_{\text{orig}}, \mathcal{B}_{\text{orig}}, P, N \rangle_R$

 1: **procedure** QX($\langle \mathcal{K}_{\text{orig}}, \mathcal{B}_{\text{orig}}, P, N \rangle_R$)
 2:     **if** ISKBVALID($\mathcal{K}_{\text{orig}}, (\mathcal{B}_{\text{orig}}, P, N, R)$) **then**
 3:         **return** 'no conflict'
 4:     **else if** $\mathcal{K}_{\text{orig}} = \emptyset$ **then**
 5:         **return** $\emptyset$
 6:     **else**
 7:         **return** QX'($\emptyset, \langle \mathcal{K}_{\text{orig}}, \mathcal{B}_{\text{orig}}, P, N \rangle_R$)

 8: **procedure** QX'($\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$)
 9:     **if** $\mathcal{C} \neq \emptyset \wedge \neg$ISKBVALID($\mathcal{B}, \langle \cdot, \emptyset, P, N \rangle_R$) **then**
10:         **return** $\emptyset$
11:     **if** $|\mathcal{K}| = 1$ **then**
12:         **return** $\mathcal{K}$
13:     $k \leftarrow$ SPLIT($|\mathcal{K}|$)
14:     $\mathcal{K}_1 \leftarrow$ GET($\mathcal{K}, 1, k$)
15:     $\mathcal{K}_2 \leftarrow$ GET($\mathcal{K}, k+1, |\mathcal{K}|$)
16:     $\mathcal{C}_2 \leftarrow$ QX'($\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R$)
17:     $\mathcal{C}_1 \leftarrow$ QX'($\mathcal{C}_2, \langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$)
18:     **return** $\mathcal{C}_1 \cup \mathcal{C}_2$

19: **procedure** ISKBVALID($\mathcal{K}, \langle \cdot, \mathcal{B}, P, N \rangle_R$)
20:     $\mathcal{K}' \leftarrow \mathcal{K} \cup \mathcal{B} \cup \bigcup_{p \in P} p$
21:     **if** $\neg$VERIFYREQ($\mathcal{K}', R$) **then**
22:         **return** $false$
23:     **for** $n \in N$ **do**
24:         **if** ENTAILS($\mathcal{K}', n$) **then**
25:             **return** $false$
26:     **return** $true$

---

be entailed by a solution KB w.r.t. the input DPI (the parts of the formulas relevant for the entailment to hold are underlined). Returning $\emptyset$ in this case means discarding $\mathcal{K}_2 = \{ax_4, ax_5\}$.

So, the algorithm opens a right branch from the root to node ③ by calling QX' (line 17) with parameters $\mathcal{C} = \emptyset$ (result of left branch), $\mathcal{K} = \mathcal{K}_1 = \{ax_1, ax_2, ax_3\}$ and $\mathcal{B} = \mathcal{B}$. During the execution of this call $\mathcal{K}_1$ is partitioned into $\{ax_1, ax_2\}$ (left branch to node ④) and $\{ax_3\}$ (right branch to node ⑤). In node ④, it holds that $\mathcal{B} \cup \{ax_1, ax_2\}$ can be extended to a solution KB by adding U_P, i.e. $\mathcal{B} \cup \{ax_1, ax_2\}$ is valid. As it is already an established fact since the execution of node ② that $\mathcal{B} \cup \{ax_1, ax_2, ax_3\}$ is invalid, it must be the case that ax_3 is an element of a minimal conflict set w.r.t. the input DPI (as there is a conflict set w.r.t. the input DPI in $\{ax_1, ax_2, ax_3\}$, but there is none in $\{ax_1, ax_2\}$). The algorithm accounts for that by checking whether $\mathcal{K}$ is a singleton (line 11) in which case it is guaranteed that $\mathcal{K}$ is a subset of a minimal conflict set w.r.t. the input DPI. So, node ④ returns $\{ax_3\}$. This procedure is continued until each path from the root node reaches a node where a trivial case is met. Then the recursion unwinds and, when arrived at the root node, the minimal conflict set $\langle ax_1, ax_3 \rangle$ is returned.

That $\mathcal{C} := \langle ax_1, ax_3 \rangle$ is indeed a conflict set can be recognized easily by the underlinings in the formulas given before. Minimality is given since $\mathcal{B} \cup \mathcal{C} \cup U_P$ is neither inconsistent nor incoherent and the deletion of any formula from $\mathcal{C}$ breaks the entailment of n_1. Hence, QX has returned a sound output.   □

i	ax_i	$\mathcal{K}$	$\mathcal{B}$
1	$A \sqsubseteq B$	•	
2	$B \sqsubseteq E$	•	
3	$B \sqsubseteq D \sqcap \neg \exists s.C$	•	
4	$C \sqsubseteq \neg(D \sqcup E)$	•	
5	$D \sqsubseteq \neg B$	•	
6	$A(w)$		•
7	$A(v)$		•
8	$s(v,w)$		•
i	$p_i \in P$		
1	$B(w)$		
i	$n_i \in N$		
1	$\neg C(w)$		
i	$r_i \in R$		
1	consistency		
2	coherency		

Table 4.3: Description Logic Example DPI 2

The complexity of Algorithm 1 in terms of the number of calls to the function ISKBVALID, which is the only place in the algorithm where a reasoning service is consulted, is captured by the following proposition.

Proposition 4.8 (Complexity of QX). *[Jun04] Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI and the function* SPLIT *(line 13 of Algorithm 1) be defined as* $\text{SPLIT}(n) = \lfloor \frac{n}{2} \rfloor$ *where n is a natural number. Then, the worst case number of calls to* ISKBVALID *during one call to* $\text{QX}(\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ *is in* $O(|\mathcal{C}| \log \frac{|\mathcal{K}|}{|\mathcal{C}|})$ *where $\mathcal{C}$ is the output of* $\text{QX}(\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$.

For any other definition of the function SPLIT, *the worst case number of* ISKBVALID *invocations gets larger.*

4.4.2 Correctness of Conflict Set Computation

This section is dedicated to the proof of correctness of Algorithm 1. First, we show some essential properties of QX by various Lemmata which will finally be exploited to demonstrate the overall soundness of QX.

The QX algorithm accepts a DPI $\langle \mathcal{K}_{\text{orig}}, \mathcal{B}_{\text{orig}}, P, N \rangle_R$ over some monotonic language $\mathcal{L}$ as input and returns a minimal conflict set $\mathcal{C} \subseteq \mathcal{K}_{\text{orig}}$ w.r.t. $\langle \mathcal{K}_{\text{orig}}, \mathcal{B}_{\text{orig}}, P, N \rangle_R$ as output. First, the algorithm checks whether $\mathcal{K}_{\text{orig}}$ is a valid KB w.r.t. the input DPI $\langle \cdot, \mathcal{B}_{\text{orig}}, P, N \rangle_R$ (line 2). If so, there is no conflict set for the DPI by Proposition 4.1 and the algorithm returns 'no conflict'. Otherwise, the test $\mathcal{K}_{\text{orig}} = \emptyset$ is performed (line 4). If so, then the negative outcome of the validity test executed in line 2 actually means that one of the two criteria of Proposition 3.4 is violated which, by Definition 3.6, implies that the DPI is not admissible. Invalidity of $\mathcal{K}_{\text{orig}}$ w.r.t. $\langle \cdot, \mathcal{B}_{\text{orig}}, P, N \rangle_R$ and non-admissiblity of $\langle \mathcal{K}_{\text{orig}}, \mathcal{B}_{\text{orig}}, P, N \rangle_R$ mean that there is only one minimal conflict set $\mathcal{C} = \emptyset$ by Proposition 4.2. Thus, $\emptyset$ is returned in line 5.

Lemma 4.1. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be an admissible DPI and $\mathcal{K}$ be invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. Then, there is a minimal conflict set $\mathcal{C} \supset \emptyset$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Proof. The proposition is a direct consequence of Proposition 4.2. □
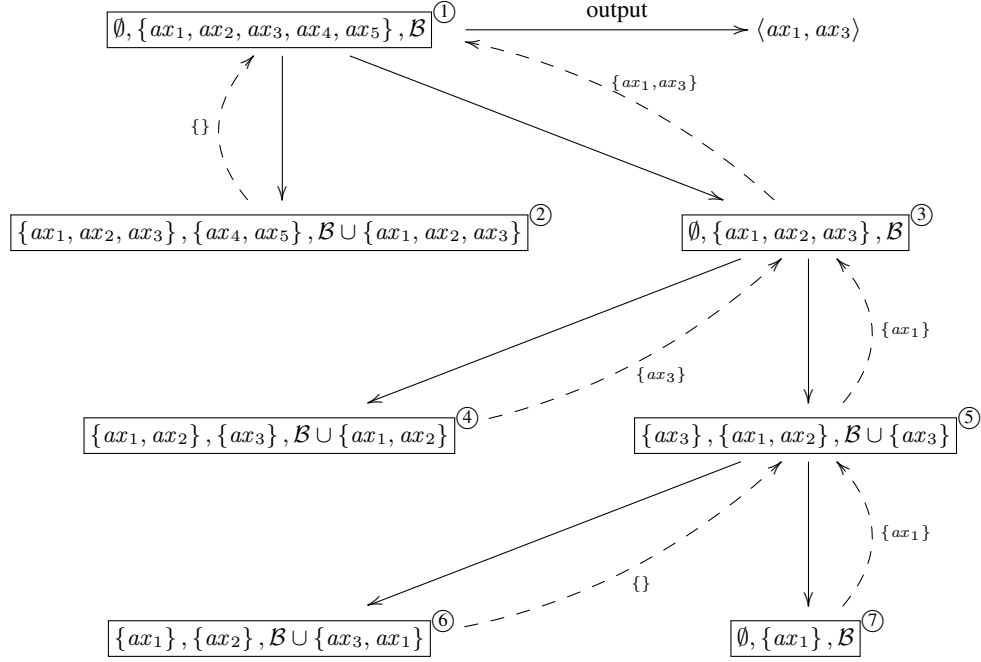
Figure 4.1: Recursion tree produced during the computation of the minimal conflict set $\langle ax_1, ax_3 \rangle$ w.r.t. the DPI shown by Table 4.3 using Algorithm 1. Nodes in the depicted tree represent calls $QX'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ and are written in format $\boxed{\mathcal{C}, \mathcal{K}, \mathcal{B}}^{(k)}$ where k is a counter starting from 1 that indicates when the respective call is made. A recursive call to QX' (left branch = call in line 16; right branch = call in line 17) is denoted by a normal arrow whereas the return of a set is visualized by a dashed arrow.

So, if both initial tests (lines 2 and 4) are negative, then, by Lemma 4.1, there is a non-trivial minimal conflict set w.r.t. $\langle \mathcal{K}_{\mathsf{orig}}, \mathcal{B}_{\mathsf{orig}}, P, N \rangle_R$ wherefore the algorithm enters the recursion by a call to the procedure QX'.

The argumentation so far proves the following lemma.

Lemma 4.2.

- $QX(\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ *returns 'no conflict' iff there is no (minimal) conflict w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

- $QX(\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ *returns* $\emptyset$ *iff* $\emptyset$ *is the only (minimal) conflict w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

- $QX(\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ *returns* $QX'(\emptyset, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ *iff there is some minimal conflict* $\mathcal{C} \supset \emptyset$ *w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Corollary 4.3. $QX(\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ *returns* $QX'(\emptyset, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ *iff* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *is an admissible DPI.*

Proof. By the third proposition of Lemma 4.2 and Proposition 4.1 we have that $QX(\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ returns $QX'(\emptyset, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ iff $\mathcal{K}$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. By Proposition 4.2, we can then conclude that $QX(\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ returns $QX'(\emptyset, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ iff $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is an admissible DPI. $\square$

The input arguments (at any call) to QX' are (a) some subset $\mathcal{C}$ of the original input KB $\mathcal{K}_{\mathsf{orig}}$ to QX and (b) a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ where $\mathcal{K} \subseteq \mathcal{K}_{\mathsf{orig}}$ and $\mathcal{B} \supseteq \mathcal{B}_{\mathsf{orig}}$.

The principle of QX' relies on the following fact.

Lemma 4.3. *[Jun04] Let $\mathcal{K}_1, \mathcal{K}_2$ be a partition of $\mathcal{K}$. If $\mathcal{C}_2$ is a minimal conflict set w.r.t. $\langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R$ and $\mathcal{C}_1$ is a minimal conflict set w.r.t. $\langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$, then $\mathcal{C}_1 \cup \mathcal{C}_2$ is a minimal conflict set w.r.t. $\langle \mathcal{K}_1 \cup \mathcal{K}_2, \mathcal{B}, P, N \rangle_R = \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Proof. Since $\mathcal{C}_1$ is a minimal conflict set w.r.t. $\langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$, we have that $\mathcal{C}_1$ is invalid w.r.t. $\langle \cdot, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$. From that we obtain that $\mathcal{C}_1 \cup \mathcal{C}_2$ must be invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. Further on, by the fact that $\mathcal{K}_1, \mathcal{K}_2$ partition $\mathcal{K}$ we have that $\mathcal{C}_1 \subseteq \mathcal{K}_1 \subseteq \mathcal{K}$ since $\mathcal{C}_1$ is a minimal conflict set w.r.t. $\langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$ and $\mathcal{C}_2 \subseteq \mathcal{K}_2 \subseteq \mathcal{K}$ since $\mathcal{C}_2$ is a minimal conflict set w.r.t. $\langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R$. Consequently, $\mathcal{C}_1 \cup \mathcal{C}_2 \subseteq \mathcal{K}$ must be true. So, by Corollary 4.1, $\mathcal{C}_1 \cup \mathcal{C}_2$ is a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

To show the minimality of $\mathcal{C}_1 \cup \mathcal{C}_2$, assume that $\mathcal{C} \subset \mathcal{C}_1 \cup \mathcal{C}_2$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Due to $\mathcal{K}_1 \cap \mathcal{K}_2 = \emptyset$ and $\mathcal{C}_1 \subseteq \mathcal{K}_1$ and $\mathcal{C}_2 \subseteq \mathcal{K}_2$, it must hold that $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$. Thus, (1) $\mathcal{C} \cap \mathcal{C}_1 \subset \mathcal{C}_1$ or (2) $\mathcal{C} \cap \mathcal{C}_2 \subset \mathcal{C}_2$.

Let us assume (1) holds. Then, $\mathcal{C}$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, i.e. $\mathcal{C} \cup \mathcal{B} \cup U_P = (\mathcal{C}_1' \cup \mathcal{C}_2) \cup \mathcal{B} \cup U_P = \mathcal{C}_1' \cup (\mathcal{B} \cup \mathcal{C}_2) \cup U_P$ violates some $r \in R$ or some $n \in N$ where $\mathcal{C}_1' \subset \mathcal{C}_1$. This, however, is a contradiction to the minimality of the conflict set $\mathcal{C}_1$ w.r.t. $\langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$.

Now, let us assume (2) holds. Then, $\mathcal{C}$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, i.e. $\mathcal{C} \cup \mathcal{B} \cup U_P = (\mathcal{C}_1 \cup \mathcal{C}_2') \cup \mathcal{B} \cup U_P$ violates some $r \in R$ or some $n \in N$ where $\mathcal{C}_2' \subset \mathcal{C}_2$. By monotonicity of $\mathcal{L}$ and $\mathcal{C}_1 \subseteq \mathcal{K}_1$, this implies $\mathcal{C}_2' \cup (\mathcal{K}_1 \cup \mathcal{B}) \cup U_P$ violates some $r \in R$ or some $n \in N$, i.e. $\mathcal{C}_2' \subset \mathcal{K}_2$ is a conflict set w.r.t. $\langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R$ which is a contradiction due to $\mathcal{C}_2' \subset \mathcal{C}_2$ and the minimality of the conflict set $\mathcal{C}_2$ w.r.t. $\langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R$. $\square$

$\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ computes a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ in a divide-and-conquer fashion whereby the argument $\mathcal{C}$ is the set of sentences of $\mathcal{K}_{\mathrm{orig}}$ that has been added to $\mathcal{B}$ in the current iteration. That is, in this iteration QX' will output either (1) $\emptyset$ if the current $\mathcal{B}$ (which includes $\mathcal{C}$) already contains a minimal conflict set w.r.t. the original DPI $\langle \mathcal{K}_{\mathrm{orig}}, \mathcal{B}_{\mathrm{orig}}, P, N \rangle_R$ or (2) a minimal conflict set w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ (i.e. a subset of a minimal conflict set w.r.t. the original DPI) which does not include any sentence from $\mathcal{C}$.

Lemma 4.4.

1. *For each call $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ within Algorithm 1 it holds that $\mathcal{C} \subseteq \mathcal{B}$.*

2. *If $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ is called in line 16 of Algorithm 1, $\mathcal{C} \neq \emptyset$ holds.*

3. *If $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ returns $\emptyset$, then there is some non-empty minimal conflict set w.r.t. $\langle \mathcal{C}, \mathcal{B} \setminus \mathcal{C}, P, N \rangle_R$.*

4. *If $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ returns $\emptyset$, then $\emptyset$ is the only minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

5. *$\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ terminates.*

Proof.

1): There are three situations when $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ is called within Algorithm 1, namely in lines 7, 16 and 17. In line 7, $\mathcal{C} := \emptyset \subseteq \mathcal{B}$ holds. In line 16, $\mathcal{C} := \mathcal{K}_1 \subseteq \mathcal{B} \cup \mathcal{K}_1 =: \mathcal{B}$ holds. In line 17, $\mathcal{C} := \mathcal{C}_2 \subseteq \mathcal{B} \cup \mathcal{C}_2 =: \mathcal{B}$ holds.

2): In line 16, QX' is called with $\mathcal{C} := \mathcal{K}_1$, which is always not the empty set due to the definition of the SPLIT function in line 13 that is used to extract $\mathcal{K}_1$ from $\mathcal{K}$.

3): The first observation is that $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ cannot return $\emptyset$ if $\mathcal{C} = \emptyset$ as in this case the first condition in line 9 is not met. Thus, in particular, QX' cannot return $\emptyset$ if called in line 7.

So, $\emptyset$ can be returned by $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ only if it is called (1) in line 16 or (2) in line 17.

If $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ returns $\emptyset$, then $\mathcal{C} \neq \emptyset$ and $\mathcal{B}$ is invalid w.r.t. $\langle \cdot, \emptyset, P, N \rangle_R$ (line 9), i.e. $\mathcal{B}$ contains a minimal conflict set w.r.t. $\langle \mathcal{B}, \emptyset, P, N \rangle_R$ which is non-empty by Proposition 4.2 since $\langle \mathcal{B}, \emptyset, P, N \rangle_R$ is an admissible DPI by admissibility of the input DPI and the invariance of P, N, R throughout QX'. Additionally, $\mathcal{C} \subseteq \mathcal{B}$ holds by the first proposition of this lemma. Now, assume that there is no non-empty (minimal) conflict set w.r.t. $\langle \mathcal{C}, \mathcal{B} \setminus \mathcal{C}, P, N \rangle_R$. Then, for each minimal conflict set $\mathcal{C}'$ (which we know is non-empty) w.r.t. $\langle \mathcal{B}, \emptyset, P, N \rangle_R$ it must hold that $\mathcal{C} \cap \mathcal{C}' = \emptyset$, i.e. there is already a non-empty minimal conflict set w.r.t. $\langle \mathcal{B} \setminus \mathcal{C}, \emptyset, P, N \rangle_R$.

Case (1): Let us assume first that the call to QX' was made in line 16. Then, before this call to QX', $\mathcal{B}$ was exactly $\mathcal{B} \setminus \mathcal{C}$. By the second proposition of this lemma, $\mathcal{C} \neq \emptyset$ as QX' was called in line 16. Thus, before the current call to QX', the algorithm must have already returned $\emptyset$ (both conditions in line 9 are met) in line 10 which is a contradiction to the assumption that $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ was called in line 16.

Case (2): Now, assume that the call to $\mathrm{QX}'(\mathcal{C}_2, \langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R)$ was made in line 17. Then $\mathcal{C}_2$ is the result of the call to $\mathrm{QX}'(\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R)$ in line 16. By the argumentation above, we have that $\mathcal{C}_2 \neq \emptyset$ and there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{B} \cup \mathcal{C}_2, \emptyset, P, N \rangle_R$. Moreover, we have that there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{B}, \emptyset, P, N \rangle_R$. However, as $\mathrm{QX}'(\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R)$ in line 16 did not return $\emptyset$ and $\mathcal{K}_1 \neq \emptyset$ by the second proposition of this lemma, it must hold that $\mathcal{B} \cup \mathcal{K}_1$ is valid w.r.t. $\langle \cdot, \emptyset, P, N \rangle_R$, i.e. there is no (minimal) conflict set w.r.t. $\langle \mathcal{B} \cup \mathcal{K}_1, \emptyset, P, N \rangle_R$. By monotonicity of $\mathcal{L}$, this is a contradiction to the fact that there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{B}, \emptyset, P, N \rangle_R$.

4): Assume $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ returns $\emptyset$ and there is some non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Since $\emptyset$ is returned, both conditions in line 2 must be met, i.e. in particular $\mathcal{B}$ must be invalid w.r.t. $\langle \cdot, \emptyset, P, N \rangle_R$ which means that $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is not admissible. By Proposition 4.2, there cannot be a non-empty (minimal) conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. This yields a contradiction.

5): $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ either returns $\emptyset$ in line 10 iff the conditions in line 9 are met or otherwise returns $\mathcal{K}$ in line 12 iff $|\mathcal{K}| = 1$ or otherwise calls itself recursively in lines 16 and 17. However, for each recursive call $\mathrm{QX}'(\mathcal{C}', \langle \mathcal{K}', \mathcal{B}', P, N \rangle_R)$ within $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ it holds that $\mathcal{K}' \subset \mathcal{K}$ as $\mathcal{K}' \in \{\mathcal{K}_1, \mathcal{K}_2\}$ and $\mathcal{K}_1, \mathcal{K}_2 \subset \mathcal{K}$ due to the definition of the SPLIT function in line 13 that is used to compute $\mathcal{K}_1$ and $\mathcal{K}_2$ from $\mathcal{K}$ in lines 14 and 15. Hence, each recursive call must finally reach the stopping criterion $|\mathcal{K}| = 1$ and return $\mathcal{K}$ if it does not reach the stopping criterion in line 9 before. $\qquad \square$

Lemma 4.5. *Let* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *be an admissible DPI. If* $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ *is called, then at least one of the immediate recursive calls of* QX' *in line 16 or line 17 is given an admissible DPI as argument.*

Proof. Let us assume that $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is an admissible DPI. Within $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$, the immediate recursive call is $\mathrm{QX}'(\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R)$ in line 16 and $\mathrm{QX}'(\mathcal{C}_2, \langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R)$ in line 17 where $\mathcal{K}_1, \mathcal{K}_2$ is a partition of $\mathcal{K}$ and $\mathcal{C}_2$ is the result of $\mathrm{QX}'(\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R)$. If $\langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R$ is admissible, then the proposition of the lemma is fulfilled. So, assume that that $\langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R$ is not admissible. Due to this non-admissibility, it must hold that $\mathcal{B} \cup \mathcal{K}_1$ is invalid w.r.t. $\langle \cdot, \emptyset, P, N \rangle_R$, so the second condition in line 2 is met. As the call to $\mathrm{QX}'(\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R)$ was made in line 16, it must be true by Lemma 4.4, prop. 2 that $\mathcal{K}_1 \neq \emptyset$ wherefore the first condition in line 2 is met as well. Thus, the result of the call of QX' in line 16 must be $\emptyset$. So, the call of QX' in line 17 looks like $\mathrm{QX}'(\emptyset, \langle \mathcal{K}_1, \mathcal{B}, P, N \rangle_R)$. However, the DPIs $\langle \mathcal{K}_1, \mathcal{B}, P, N \rangle_R$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ are identical except for the first entries, i.e. $\mathcal{K}_1$ and $\mathcal{K}$. We know that the latter DPI is admissible. Due to the fact that admissibility of a DPI is defined independently of the KB (the first entry of the DPI tuple), we have that $\langle \mathcal{K}_1, \mathcal{B}, P, N \rangle_R$ must be admissible. This completes the proof. $\qquad \square$

As long as the algorithm goes downwards in the recursion tree (and has never gone upwards), (1) the invariant that a minimal conflict set exists for each recursive call to QX' holds, (2) each call to QX' that

returns, returns a singleton or empty set and (3) the two calls to QX' immediately before going upwards in the recursion tree for the first time must both return either a singleton or an empty set.

Lemma 4.6 (QX: Downwards Correctness). *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be an admissible DPI and let there be a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then, the following propositions hold:*

1. *Before line 18 has ever been reached during the execution of $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$, the following holds: If some call to $\mathrm{QX}'(\mathcal{C}', \langle \mathcal{K}', \mathcal{B}', P, N|_R)$ returns a set S, then $S = \emptyset$ or $|S| = 1$.*

2. *Before line 18 has ever been reached during the execution of $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$, the following holds: If $\mathrm{QX}'(\mathcal{C}', \langle \mathcal{K}', \mathcal{B}', P, N \rangle_R)$ is recursively called, then there is some non-empty minimal conflict set w.r.t. $\langle \mathcal{K}' \cup \mathcal{C}', \mathcal{B}' \setminus \mathcal{C}', P, N \rangle_R$.*

3. *Before line 18 has ever been reached during the execution of $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$, the following holds: If some call to $\mathrm{QX}'(\mathcal{C}', \langle \mathcal{K}', \mathcal{B}', P, N \rangle_R)$ returns a set S, then S is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

4. *When line 18 is reached for the first time, each of the calls to QX' immediately before in lines 16 and 17 must have returned $\emptyset$ or some $\mathcal{K}$ with $|\mathcal{K}| = 1$.*

Proof.

1): Assume the opposite, i.e. some call to $\mathrm{QX}'(\mathcal{C}', \langle \mathcal{K}', \mathcal{B}', P, N \rangle_R)$ returns a set S with $|S| > 1$ before line 18 has ever been reached. There are three places where QX' can return, namely in line 10, in line 12 or in line 18. However, in line 10, only $\emptyset$ and in line 12 only a singleton set can be returned. That is, S must be returned in line 18 which is a contradiction to the assumption that line 18 has not yet been reached.

2): *Induction Base:* The first recursive call $\mathrm{QX}'(\mathcal{C}', \langle \mathcal{K}', \mathcal{B}', P, N \rangle_R)$ can only occur at line 16 where $\mathcal{C}' = \mathcal{K}_1$, $\mathcal{K}' = \mathcal{K}_2$ and $\mathcal{B}' = \mathcal{B} \cup \mathcal{K}_1$ and $\mathcal{K}_1, \mathcal{K}_2$ is a partition of $\mathcal{K}$ as per the definition of the SPLIT and GET functions in lines 13-15. So, $\mathcal{K}' \cup \mathcal{C}' = \mathcal{K}$ and $\mathcal{B}' \setminus \mathcal{C}' = \mathcal{B}$. The latter holds since $\mathcal{C}' \subseteq \mathcal{K}$ and for each DPI $\mathcal{K} \cap \mathcal{B} = \emptyset$ holds by Definition 3.1. As there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ we have that there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}' \cup \mathcal{C}', \mathcal{B}' \setminus \mathcal{C}', P, N \rangle_R$ by the fact that $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R = \langle \mathcal{K}' \cup \mathcal{C}', \mathcal{B}' \setminus \mathcal{C}', P, N \rangle_R$. Thus, the existence of a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}' \cup \mathcal{C}', \mathcal{B}' \setminus \mathcal{C}', P, N \rangle_R$ is given during the execution of the first recursive call to QX'.

Induction Assumption: Now, let us assume that the existence of a non-empty minimal conflict set w.r.t. $\langle \mathcal{K} \cup \mathcal{C}, \mathcal{B} \setminus \mathcal{C}, P, N \rangle_R$ is given during some call $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$. The goal is now to show that the existence of a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}' \cup \mathcal{C}', \mathcal{B}' \setminus \mathcal{C}', P, N \rangle_R$ is given during any recursive call $\mathrm{QX}'(\mathcal{C}', \langle \mathcal{K}', \mathcal{B}', P, N \rangle_R)$ that is invoked during execution of $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$.

Induction Step: Now, there are three cases where this recursive call to QX' can take place, namely (1) in line 16, (2) in line 17 where the result of QX' in line 16 is $\mathcal{C}_2 = \emptyset$ and (3) in line 17 where the result of QX' in line 16 is some $\mathcal{C}_2$ with $|\mathcal{C}_2| = 1$. The case where some $\mathcal{C}_2$ with $|\mathcal{C}_2| > 1$ is returned by QX' in line 16, is impossible due to the assumption that line 18 has not yet been reached and the first proposition of this lemma.

Case (1): Let us assume that the call $\mathrm{QX}'(\mathcal{C}', \langle \mathcal{K}', \mathcal{B}', P, N \rangle_R)$ is made in line 16. Since that call is made within $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$, it must hold that some condition in line 2 during $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ is violated, as otherwise a return would have taken place in line 10 which is a contradiction to the assumption that $\mathrm{QX}'(\mathcal{C}', \langle \mathcal{K}', \mathcal{B}', P, N \rangle_R)$ is called in line 16.

Let us first assume that $\mathcal{C} = \emptyset$ holds. In this case, the first condition in line 2 is violated and, by the *Induction Assumption*, it is true that there is a non-empty minimal conflict set w.r.t. the DPI $\langle \mathcal{K} \cup \mathcal{C}, \mathcal{B} \setminus \mathcal{C}, P, N \rangle_R$ which is equal to the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ by $\mathcal{C} = \emptyset$. So, an equal argumentation to the one of the *Induction Base* can be applied to derive that there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}' \cup \mathcal{C}', \mathcal{B}' \setminus \mathcal{C}', P, N \rangle_R$.

If $\mathcal{C} \neq \emptyset$ holds, on the other hand, then the first condition in line 2 is satisfied wherefore the second condition in line 2 must be violated. That is, there is no conflict set w.r.t. $\langle \mathcal{B}, \emptyset, P, N \rangle_R$. As there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{K} \cup \mathcal{C}, \mathcal{B} \setminus \mathcal{C}, P, N \rangle_R$ by the *Induction Assumption*, $\mathcal{C} \subseteq \mathcal{B}$ by Lemma 4.4, prop. 1 and $|\mathcal{K}| \geq 2$ by the fact that there was no return in line 12, there must be a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Again, an equal argumentation to the one of the *Induction Base* can be applied to derive that there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}' \cup \mathcal{C}', \mathcal{B}' \setminus \mathcal{C}', P, N \rangle_R$.

Case (2): Here, we assume that the recursive call $\mathrm{QX}'(\mathcal{C}', \langle \mathcal{K}', \mathcal{B}', P, N \rangle_R)$ is made in line 17 and the result of QX' in line 16 is $\mathcal{C}_2 = \emptyset$. So, it holds that $\mathcal{C}' = \mathcal{C}_2 = \emptyset$, $\mathcal{K}' = \mathcal{K}_1$ and $\mathcal{B}' = \mathcal{B}$, i.e. the recursive call can be written as $\mathrm{QX}'(\emptyset, \langle \mathcal{K}_1, \mathcal{B}, P, N \rangle_R)$. By the fact that $\mathrm{QX}'(\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R)$ called in line 16 returned $\emptyset$, both conditions in line 2 during $\mathrm{QX}'(\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R)$ must have been met. Thus, in particular the existence of a non-empty minimal conflict set w.r.t. $\langle \mathcal{B} \cup \mathcal{K}_1, \emptyset, P, N \rangle_R$ must be given. Further on, by the *Induction Assumption* there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{C} \cup \mathcal{K}, \mathcal{B} \setminus \mathcal{C}, P, N \rangle_R$.

Let us first assume $\mathcal{C} = \emptyset$. In this case $\langle \mathcal{C} \cup \mathcal{K}, \mathcal{B} \setminus \mathcal{C}, P, N \rangle_R$ can be written as $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and it holds that there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, i.e. $\mathcal{K}$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. By Proposition 4.2, this implies that $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is admissible. In other words, there is no conflict set w.r.t. $\langle \mathcal{B}, \emptyset, P, N \rangle_R$. Consequently, there must be a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}_1, \mathcal{B}, P, N \rangle_R$.

If $\mathcal{C} \neq \emptyset$, on the other hand, then the second condition in line 2 during $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ must be invalid, i.e. there is no conflict set w.r.t. $\langle \mathcal{B}, \emptyset, P, N \rangle_R$. Consequently, there must be a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}_1, \mathcal{B}, P, N \rangle_R$.

Case (3): Here, we assume that the recursive call $\mathrm{QX}'(\mathcal{C}', \langle \mathcal{K}', \mathcal{B}', P, N \rangle_R)$ is made in line 17 and the result of QX' in line 16 is $\mathcal{C}_2 \neq \emptyset$. As $\mathcal{C}_2 \neq \emptyset$ and line 18 has never been reached by assumption, $\mathcal{C}_2$ must have been returned in line 12 of $\mathrm{QX}'(\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R)$ (which was called in line 16) wherefore $\mathcal{C}_2 = \mathcal{K}_2$ must hold. So, it holds that $\mathcal{C}' = \mathcal{K}_2$, $\mathcal{K}' = \mathcal{K}_1$ and $\mathcal{B}' = \mathcal{B} \cup \mathcal{K}_2$, i.e. the recursive call can be written as $\mathrm{QX}'(\mathcal{K}_2, \langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{K}_2, P, N \rangle_R)$. By the *Induction Assumption*, there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{C} \cup \mathcal{K}, \mathcal{B} \setminus \mathcal{C}, P, N \rangle_R$. Moreover, $\mathcal{C} \subseteq \mathcal{B}$ by Lemma 4.4, prop. 1 and (*) there is a non-empty minimal conflict set w.r.t. the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ which is equal to the DPI $\langle \mathcal{K}_1 \cup \mathcal{K}_2, \mathcal{B}, P, N \rangle_R$ by the fact that $\mathcal{K}_1, \mathcal{K}_2$ partition $\mathcal{K}$ as per the definition of the SPLIT and GET functions in lines 13-15.

What must still be proven, is (*): Let us first assume that $\mathcal{C} = \emptyset$ holds. In this case, $\langle \mathcal{C} \cup \mathcal{K}, \mathcal{B} \setminus \mathcal{C}, P, N \rangle_R = \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and thus there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

If $\mathcal{C} \neq \emptyset$, on the other hand, then the second condition in line 2 during $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ must be invalid as otherwise $\emptyset$ would have been returned which is a contradiction to the assumption that the recursive call $\mathrm{QX}'(\mathcal{C}', \langle \mathcal{K}', \mathcal{B}', P, N \rangle_R)$ was invoked in line 17. So, there is no conflict set w.r.t. $\langle \mathcal{B}, \emptyset, P, N \rangle_R$. Consequently, there must be a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ due to $\mathcal{C} \subseteq \mathcal{B}$ by Lemma 4.4, prop. 1.

3): Case $S \neq \emptyset$: By $S \neq \emptyset$ and the fact that line 18 has not yet been reached, we obtain by the first proposition of this lemma that $|S| = 1$ must hold.

There are two cases that can trigger $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ to return $\mathcal{K}$ with $|\mathcal{K}| = 1$, i.e. case 1 involving $\mathcal{C} \neq \emptyset$ and case 2 involving $\mathcal{C} = \emptyset$.

In case 1, $\mathcal{B}$ must be valid w.r.t. $\langle \cdot, \emptyset, P, N, \rangle_R$ as otherwise $\emptyset$ would be returned in line 10. So, there is no (minimal) conflict set w.r.t. $\langle \mathcal{B}, \emptyset, P, N \rangle_R$.

As $|\mathcal{K}| = 1$ by assumption and by the fact that $\mathcal{C} \subseteq \mathcal{B}$ (holds by Lemma 4.4, prop. 1) and there is some non-empty minimal conflict set w.r.t. $\langle \mathcal{K} \cup \mathcal{C}, \mathcal{B} \setminus \mathcal{C}, P, N \rangle_R$ (holds by the second proposition of this lemma), $\mathcal{K}$ must include a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Since the only proper subset of $\mathcal{K}$ is the empty set, $\mathcal{K}$ must be a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Case 2 can arise only when $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ is called in line 7 or line 17. In line 16 QX' is called with $\mathcal{C} \neq \emptyset$ by Lemma 4.4, prop. 2.

In line 7 QX' is called with $\mathcal{C} = \emptyset$ and, by Corollary 4.3, with an admissible DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ for

which a non-empty minimal conflict set exists as arguments. By the second proposition of this lemma, there is some non-empty minimal conflict set w.r.t. $\langle \mathcal{K} \cup \emptyset, \mathcal{B} \setminus \emptyset, P, N \rangle_R = \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, and, by admissibility of $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, there is no (minimal) conflict set w.r.t. $\langle \mathcal{B}, \emptyset, P, N \rangle_R$. By $|\mathcal{K}| = 1$, $\mathcal{K}$ must be a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

A necessary condition for QX′ to be called with $\mathcal{C} = \emptyset$ in line 17 is obviously that QX′$(\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R)$ called in line 16 returns $\emptyset$. By the Lemma 4.4, prop. 3, there is some non-empty minimal conflict set w.r.t. $\langle \mathcal{K}_1, \mathcal{B}, P, N \rangle_R$. In line 17, the call QX′$(\emptyset, \langle \mathcal{K}_1, \mathcal{B}, P, N \rangle_R)$ is made which, by assumption, returns $\mathcal{K}_1$ with $|\mathcal{K}_1| = 1$. That means $\mathcal{K}_1$ is a minimal conflict set w.r.t. $\langle \mathcal{K}_1, \mathcal{B}, P, N \rangle_R$.

Case $S = \emptyset$: Here, both conditions in line 2 must be met, i.e. in particular $\mathcal{B}$ is invalid w.r.t. $\langle \cdot, \emptyset, P, N \rangle_R$ which implies that $\mathcal{K}$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is admissible. Therefore, by Proposition 4.2, there is no non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. However, since $\mathcal{K}$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, there must be a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. So, there is only the empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

4): This proposition is an immediate consequence of the first proposition of this lemma. □

Lemma 4.7. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a non-admissible DPI. Then, $\emptyset$ is the only minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and* QX′$(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ *with $\mathcal{C} \neq \emptyset$ returns $\emptyset$ immediately in line 10.*

Proof. Since $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is non-admissible, $\mathcal{B} \cup U_P$ violates some $r \in R$ or $\mathcal{B} \cup U_P \models n$ for some $n \in N$. Therefore, $\emptyset$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, which, by Corollary 4.1, implies that $\emptyset$ is a (minimal) conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

QX′$(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ returns $\emptyset$ in line 10 as both conditions in line 9 are satisfied due to $\mathcal{C} \neq \emptyset$ and the non-admissibility of $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. □

Lemma 4.8. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be an admissible DPI. Then* QX′$(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ *does not return in line 10.*

Proof. By Definition 3.6, $\mathcal{B}$ must be valid w.r.t. $\langle \cdot, \emptyset, P, N \rangle_R$. Hence, the second condition in line 9 is not satisfied wherefore a return cannot take place in line 10. □

Lemma 4.9. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be an admissible DPI and let there be a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then the following holds: When* QX′$(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ *reaches line 18 for the first time, $\mathcal{C}_1 \cup \mathcal{C}_2$ is a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Proof. The premises of this lemma are the same as those of Lemma 4.6. By Lemma 4.6, prop. 4 we know that for $\mathcal{C}_2$ and $\mathcal{C}_1$ that are returned by the the calls to QX′ in lines 16 and 17 $|\mathcal{C}_1| \leq 1$ and $|\mathcal{C}_2| \leq 1$ holds. Moreover, we know by Lemma 4.3 that $\mathcal{C}_1 \cup \mathcal{C}_2$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

What remains open is to show that $\mathcal{C}_1 \cup \mathcal{C}_2 \neq \emptyset$. To this end, we first assume that $\mathcal{C} \neq \emptyset$. Then, by Lemma 4.7, $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ must be an admissible DPI since it does not return in line 10, but only in line 18.

If, on the other hand, $\mathcal{C} = \emptyset$ holds, we can apply Lemma 4.6, prop. 2 to obtain that there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. This implies that $\mathcal{K}$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. Therefore, we can conclude by means of Proposition 4.2 that $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is an admissible DPI.

Thus, in both cases we have that $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is an admissible DPI. Applying Lemma 4.5 yields that at least one recursive call to QX′ in lines 16 and 17 is given an admissible DPI as argument. By Lemma 4.8, this call cannot return in line 10. So, it must return in line 12 by the assumption that line 18 has not yet been reached before, wherefore it must return a set of cardinality 1. This completes the proof. □

As long as the algorithm goes upwards after going upwards for the first time, a non-empty minimal conflict set is propagated upwards.

Lemma 4.10 (QX: Upwards Correctness). *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be an admissible DPI and let there be a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then: After $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ has reached line 18 for the first time, the following holds: As long as line 16 is not reached, each return in line 18 returns a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Proof. The premises of this lemma are the same as those of Lemma 4.6. By Lemma 4.9 we know that a non-empty minimal conflict $\mathcal{C}$ set is returned at the first return that is made in line 18. As, by assumption, $\mathcal{C}$ is not the result $\mathcal{C}_2$ of a prior call to QX' in line 16, it must be the result $\mathcal{C}_1$ of a prior call to QX' in line 17. Since the premises of Lemma 4.6 are fulfilled, Lemma 4.6 can be applied. Since the call $\mathrm{QX}'(\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle)$ (that returned $\mathcal{C}_2$) in line 16 took place before line 18 was first reached, we have that $\mathcal{C}_2$ is a minimal conflict set w.r.t. $\langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle$ by Lemma 4.6, prop. 3. By Lemma 4.3, we have that $\mathcal{C}_2 \cup \mathcal{C}$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$. As long as line 16 is not reached, the same argumentation can be used to show that a minimal conflict set is returned in line 18. $\square$

When the algorithm goes downwards again after going upwards for the first time, the invariant that that a minimal conflict set exists for each recursive downwards call to QX' holds.

Lemma 4.11 (QX: Downwards-after-upwards Correctness). *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be an admissible DPI and let there be a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then: After $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ has reached line 18 for the first time, the following holds: If line 16 is reached for the first time, then, if the DPI $\langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$ which is the argument to the immediate call $\mathrm{QX}'(\mathcal{C}_2, \langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R)$ in line 17 is admissible, then there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$.*

Proof. The premises of this lemma are the same as those of Lemma 4.6. Since line 16 is first reached after line 18 has been reached for the first time, it must hold that $\mathrm{QX}'(\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R)$ in line 16 was called before line 18 has been reached. The reason for this to hold is the fact that only returns and no new calls to QX' can have been made between the first occurrence of line 18 and the next occurrence of line 16.

Therefore, the result $\mathcal{C}_2$ of the call $\mathrm{QX}'(\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R)$ in line 16 is a minimal conflict set w.r.t. $\langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R$ due to Lemma 4.6, prop. 3. As a consequence, $\mathcal{C}_2 \cup \mathcal{B} \cup \mathcal{K}_1 \cup U_P$ violates some $r \in R$ or some $N \in N$. As the DPI $\langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$ is admissible by assumption, it holds that $\mathcal{C}_2 \cup \mathcal{B} \cup U_P$ does not violate any $r \in R$ or $N \in N$. Hence, $\mathcal{K}_1$ must be invalid w.r.t. $\langle \cdot, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$ which implies that there must be a non-empty minimal conflict set S w.r.t. $\langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$. $\square$

By applying the argumentation of Lemmas 4.6, 4.10 and 4.11 recursively on the entire recursion tree, we can prove the correctness of QX'.

Lemma 4.12. *If $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}_{\mathrm{orig}}, \mathcal{B}_{\mathrm{orig}}, P, N \rangle_R)$ is called in line 7 by Algorithm 1, it returns a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}_{\mathrm{orig}}, \mathcal{B}_{\mathrm{orig}}, P, N \rangle_R$.*

Proof. If $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}_{\mathrm{orig}}, \mathcal{B}_{\mathrm{orig}}, P, N \rangle_R)$ is called in line 7 of Algorithm 1, it must be true, by Lemma 4.2, prop. 4.2 and Corollary 4.3, that $\langle \mathcal{K}_{\mathrm{orig}}, \mathcal{B}_{\mathrm{orig}}, P, N \rangle_R$ is an admissible DPI for which a non-empty minimal conflict set exists. As a consequence, the premises of Lemma 4.6 are met for $\langle \mathcal{K}_{\mathrm{orig}}, \mathcal{B}_{\mathrm{orig}}, P, N \rangle_R$.

There are two cases to consider: Either (a) $|\mathcal{K}_{\mathrm{orig}}| \leq 1$ or (b) $|\mathcal{K}_{\mathrm{orig}}| > 1$ for the initial call to $\mathrm{QX}'(\mathcal{C}, \langle \mathcal{K}_{\mathrm{orig}}, \mathcal{B}_{\mathrm{orig}}, P, N \rangle_R)$ in line 7. In case (a), $0 = |\mathcal{K}_{\mathrm{orig}}| < 1$ cannot hold as there must be a non-empty minimal conflict set $\mathcal{C}$ w.r.t. $\langle \mathcal{K}_{\mathrm{orig}}, \mathcal{B}_{\mathrm{orig}}, P, N \rangle_R$ due to Lemma 4.2, prop. 4.2. Since $\emptyset \subset \mathcal{C} \subseteq \mathcal{K}_{\mathrm{orig}}$ must hold for $\mathcal{C}$, this would be a contradiction to $|\mathcal{K}_{\mathrm{orig}}| = 0$.

So, $|\mathcal{K}_{\mathrm{orig}}| = 1$ holds in case (a). In this case, QX' returns $\mathcal{K}_{\mathrm{orig}}$ immediately in line 12, since $\mathcal{C} = \emptyset$ and thus the conditions checked in line 9 cannot be met. In this case, $\mathcal{K}_{\mathrm{orig}}$ is indeed a non-empty minimal conflict set since for the DPI $\langle \mathcal{K}_{\mathrm{orig}}, \mathcal{B}_{\mathrm{orig}}, P, N \rangle_R$ given as argument there is a non-empty minimal conflict set by Lemma 4.2, prop. 4.2. Therefore $\emptyset$ cannot be a conflict set w.r.t. this DPI whereby $\mathcal{K}_{\mathrm{orig}}$ is the only possible minimal conflict set due to $|\mathcal{K}_{\mathrm{orig}}| = 1$.

Case (b): In this case, a direct return can neither take place in line 10 by $\mathcal{C} = \emptyset$ nor in line 12 by $|\mathcal{K}_{\text{orig}}| > 1$. So, QX$'$ is called recursively in lines 16 and 17. Since QX$'$ terminates due to Lemma 4.2, prop. 5, QX$'$ must reach line 18. The first time some recursive call QX$'(\mathcal{C}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ reaches line 18, it returns a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ due to Lemma 4.9.

By Lemma 4.10, as long as line 16 is not reached, i.e. no "left branch" (call to QX$'$ in line 16) but only "right branches" (calls to QX$'$ in line 17) return, a minimal conflict set S is returned for each call to QX$'$ that "wraps" (is higher in the recursion tree than) the call that was the first to reach line 18. It holds that $S \neq \emptyset$ since S is a union of sets including the non-empty set returned when line 18 was first reached.

When it comes to an execution of line 16, i.e. the left branch returns, then the algorithm will take the right branch by executing line 17, i.e. calling QX$'(\mathcal{C}_2, \langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R)$, and go downwards in the recursion tree.

Now, there are two cases. First, $\langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$ is non-admissible. Then, by Lemma 4.7, there is only one minimal conflict set w.r.t. $\langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$, namely $\emptyset$, and QX$'(\mathcal{C}_2, \langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R)$ directly returns $\emptyset$. As also the result $\mathcal{C}_2$ of the call to QX$'(\mathcal{K}_1, \langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R)$ immediately before in line 16 is a minimal conflict set w.r.t. $\langle \mathcal{K}_2, \mathcal{B} \cup \mathcal{K}_1, P, N \rangle_R$, as established above, we can apply Lemma 4.3 to derive that indeed a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is returned in line 18. Thus, Lemma 4.10 can be further applied to move upwards in the recursion tree until line 16 occurs again.

Second, $\langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$ is admissible. Then, by Lemma 4.11, there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$. Hence, Lemma 4.6 can be used again for the subtree of the recursion tree rooted at the call QX$'(\mathcal{C}_2, \langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R)$. That is, it can be used to show that each call to QX$'$ within this subtree returns a minimal conflict set w.r.t. the DPI given as argument as long as the algorithm moves downwards in the tree. Having reached line 18 for the first time, Lemma 4.9 lets us conclude again that a non-empty conflict set w.r.t. the respective argument DPI is actually returned at this place. Subsequently, Lemma 4.10 can be applied to show that each return gives back a minimal conflict set w.r.t. the argument DPI of the respective call, as long as the algorithm moves upwards in the recursion tree.

What is still open is to show that the call QX$'(\mathcal{C}_2, \langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R)$ in line 17 that is made immediately after the algorithm first reached line 16 after moving upwards after reaching line 18 for the first time returns a minimal conflict set w.r.t. $\langle \mathcal{K}_1, \mathcal{B} \cup \mathcal{C}_2, P, N \rangle_R$, indeed. This holds by the fact that Lemmas 4.6 and 4.10 guarantee that a left branch always returns a minimal conflict set, Lemma 4.11 guarantees that Lemmas 4.6 and 4.10 can be applied after making a single right branch. However, as QX$'$ terminates the recursion tree is finite and thus the case must arise where the right branch directly returns. In case the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given as argument for this right branch is non-admissible, the only minimal conflict set $\emptyset$ is returned, as established above. If the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given as argument for this right branch is admissible, on the other hand, then we have already shown above that there is a non-empty minimal conflict set w.r.t. this DPI. Moreover, $|\mathcal{K}| = 1$ must hold due to the fact that this right branch directly returns (without entering a further recursion). Therefore, $\mathcal{K}$ is returned which is actually a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ as $\mathcal{K}$ is the only non-empty subset of $\mathcal{K}$. □

Proposition 4.9. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI. Then, $\mathrm{QX}(\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ terminates and returns*

- *'no conflict' iff there is no conflict w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$*
  ($\mathcal{K}$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$)

- *$\emptyset$ iff $\emptyset$ is the only minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$*
  (DPI is non-admissible)

- *a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ iff there is a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$*
  (DPI is admissible and $\mathcal{K}$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$).

Proof. The proposition is a direct consequence of Lemma 4.2 and Lemma 4.12. □

4.5   Hitting Set Tree Based Diagnosis Computation

One way to compute minimal diagnoses from minimal conflict sets is to use a hitting set tree algorithm which was originally proposed by Reiter [Rei87]. In this work we describe methods for non-interactive and interactive diagnosis computation based on the ones used in [FS05, SF10, SFFR12] which are closely related to the original hitting set tree algorithm. Differences of the described non-interactive algorithm to the original one of Reiter are

1. the usage of different edge weights (probabilities) inducing an order of node generation (uniform-cost) different to breadth-first and

2. the opportunity to specify an execution time threshold t as well as a minimal ($n_{\min}$) and maximal ($n_{\max}$) desired number of minimal diagnoses to be computed by the algorithm.

In this vein, the algorithm computes at least the $n_{\min}$ most-probable minimal diagnoses w.r.t. the given probabilities and goes on computing further next most-probable minimal diagnoses until either overall computation time reaches the time limit t or $n_{\max}$ diagnoses have been computed.

Such a time threshold and an interval of minimal and maximal number of diagnoses is particularly relevant in settings where not all potential minimal faulty sets need to be computed, such as iterative, interactive settings where reaction time is crucial (since a user is waiting to interact with the system). Instead, in such settings only a "representative" set of minimal diagnoses is exploited to decide which question to ask a user such that the answer to that question allows the constructed partial tree to be pruned. After pruning, the tree is expanded again to compute another "representative" set of minimal diagnoses. Such an interactive KB debugging algorithm will be presented in Part II. The non-interactive version of the KB debugging algorithm is delineated by Algorithm 2 and described next.

Inputs.   The algorithm takes as input an admissible DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, some computation timeout t, a desired minimal ($n_{\min}$) and maximal ($n_{\max}$) number of minimal diagnoses to be returned, and a function $p : \mathcal{K} \to (0, 0.5)$ that assigns to each formula $ax \in \mathcal{K}$ a weight that represents the (estimated) likeliness of ax to be faulty and thereby determines the search strategy, e.g. breadth-first or uniform-cost. Within the algorithm, $p()$ is used to impose an order on open nodes that tells the algorithm which node to expand next. Details concerning the function $p()$ will be discussed in Section 4.6 after demonstrating various ways of obtaining information relevant to $p()$ and detailing how $p()$ can be defined by means of such information. Throughout the rest of the current Section 4.5 we assume that $p()$ implies a first-in-first-out sorting of open nodes, i.e. a breadth-first search strategy as described in [Rei87].

4.5.1   Breadth-First Diagnosis Computation

Algorithm Overview and Implementation Remarks.   To compute minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ from minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, the algorithm produces a labeled tree where a non-closed node is labeled by a minimal conflict set and a closed node is labeled by either *valid* or *closed*. From a non-closed node labeled by a minimal conflict set $\mathcal{C} = \{ax_p, \ldots, ax_q\}$ there are $|\mathcal{C}|$ outgoing edges, each labeled by one $ax \in \mathcal{C}$ and each leading to a new node that needs to be labeled. Closed nodes are leaf nodes of the produced tree, i.e. they have no successor nodes, and correspond to non-minimal or duplicate hitting sets (label *closed*) or to minimal hitting sets (label *valid*) of all minimal conflict sets w.r.t. the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Conflict sets to label nodes are computed only on-demand for time efficiency after the attempt to reuse an already computed one fails. In case an appropriate order of node labeling (e.g. breadth-first tree construction) is used, the complete tree given when all nodes in the tree are closed contains all minimal diagnoses w.r.t. the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ provided as input. In this complete tree, the set of edge labels on each path from the root node to a node labeled by *valid* is a minimal diagnosis.

What Algorithm 2 actually does is building up a *pruned HS-tree* for a given DPI. So, we next provide formal definitions of a *(partial) HS-tree* and a *(partial) pruned HS-tree* based on the definitions given in [Rei87].

Definition 4.7 (HS-Tree). *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be an admissible DPI. An edge-labeled and node-labeled tree T is called an* HS-tree *w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ iff it is a smallest tree with the following properties:*

1. *The root of T is labeled by $valid$ if $\mathcal{K}$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. Otherwise, the root is labeled by a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

2. *If n is a node of T, define $H(\mathsf{n})$ to be the set of edge labels on the path in T from the root node to n. If n is labeled by $valid$, it has no successor nodes in T. If n is labeled by a conflict set $\mathcal{C}$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, then for each $ax \in \mathcal{C}$, n has a successor node n_{ax} joined to n by an edge labeled by ax. The label for n_{ax} is a conflict set $\mathcal{C}'$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ such that $\mathcal{C}' \cap H(\mathsf{n}_{ax}) = \emptyset$ if such a set $\mathcal{C}'$ exists. Otherwise, n_{ax} is labeled by $valid$.*

T is called a partial HS-tree *w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ iff T is a HS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ where not all nodes in T are labeled and non-labeled nodes have no successors.*

Definition 4.8 (Pruned HS-Tree). *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be an admissible DPI. An edge-labeled and node-labeled tree T is called a* pruned HS-tree (pHS-tree) *w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ iff T is the result of constructing an HS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ with due regard to the following rules:*

1. *Label nodes in the HS-tree in* breadth-first order.

2. *Use only* minimal *conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ to label nodes in T.*

3. *Reusing node labels: If node n is labeled by $\mathcal{C}$ and n' is a node such that $H(\mathsf{n}') \cap \mathcal{C} = \emptyset$, label n' by $\mathcal{C}$.*

4. *Non-minimality pruning rule: If node n is labeled by $valid$ and node n' is such that $H(\mathsf{n}) \subseteq H(\mathsf{n}')$, label n' by $closed$.*

5. *If node n is labeled by $closed$, it has no successors.*

6. *Duplicate pruning rule: If node n is next to be labeled and there is some node n' such that $H(\mathsf{n}') = H(\mathsf{n})$, then label n by $closed$.*

T is called a partial pruned HS-tree *iff T is a pruned HS-tree where not all nodes in T have been labeled yet and non-labeled nodes have no successors.*

Remark 4.1   Notice that we use a definition of a pruned HS-tree that slightly differs from the definition given in [Rei87] in that we inherently assume that only *minimal* conflict sets w.r.t. the given DPI are used to label nodes in the tree. Therefore we could omit the last rule in the definition of [Rei87]. Namely, such a situation where some node has been labeled by a subset of the label of another node cannot arise in our definition since no minimal conflict set can be a subset of another different minimal conflict set w.r.t. the same DPI.

In general, there are multiple different pHS-trees w.r.t. one and the same DPI [GSW89]. Reason for this is that

- the order of adding successor nodes (on the same tree level) to the queue **Q** and

- which of generally multiple minimal conflict sets to (re)use to label a node

is not determined by Definition 4.8.   □

By [Rei87, Theorem 4.8] and Proposition 4.6, the following holds:

Proposition 4.10. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be an admissible DPI and T a pHS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then,* $\{H(\mathsf{n}) \mid \mathsf{n}$ *is a node of T labeled by valid*$\} = \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, *i.e. the set of all minimal diagnoses w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Remark 4.2   A *node* nd in Algorithm 2 is defined as the set of formulas that label the edges on the path from the root node to nd. In other words, we associate a node n with $H(\mathsf{n})$. In this vein, Algorithm 2 *internally* does not store a labeled tree, but only "relevant" sets of nodes and conflict sets. That is, it does not store any

- non-leaf nodes,

- labels of non-leaf nodes, i.e. it does not store which minimal conflict set labels which node,

- edges between nodes,

- labels of edges and

- leaf nodes labeled by *closed*.

Let T denote the (partial) pHS-tree produced by Algorithm 2 at some point during its execution (Corollary 4.4 will show that Algorithm 2 using breadth-first search in fact produces a (partial) pHS-tree). Then, Algorithm 2 only stores

- a set of nodes $\mathbf{D}_{calc}$ where each node corresponds to the edge labels along a path in T leading to a leaf node that has been labeled by *valid* (minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$),

- a list of open (non-closed) nodes $\mathbf{Q}$ where each node in $\mathbf{Q}$ corresponds to the edge labels along a path in T leading from the root node to a leaf node that has been generated, but has not yet been labeled and

- the set $\mathbf{C}_{calc}$ of already computed minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that have been used to label non-leaf nodes in T.

We call $\langle \mathbf{D}_{calc}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ the *relevant data of T*. If T is a pHS-tree, then $\mathbf{Q}$ is the empty list.

This internal representation of the constructed (partial) pHS-tree by its relevant data does not constrain the functionality of the algorithm. This holds as diagnoses are paths from the root, i.e. nodes in the internal representation, and the goal of a (partial) pHS-tree is to determine minimal diagnoses w.r.t. the given DPI. The node labels or edge labels along a certain path and their order along this path is completely irrelevant when it comes to finding a label for the leaf node of this path. Instead, only the set of edge labels is required for the computation of the label for a leaf node. Also, to rule out nodes corresponding to non-minimal diagnoses, it is sufficient to know the set of already found diagnoses $\mathbf{D}_{calc}$. No already closed nodes are needed for the correct functionality of Algorithm 2.   □

Initialization.   First, Algorithm 2 initializes the variable t_{start} with the current system time (GETTIME), the set of calculated minimal diagnoses $\mathbf{D}_{calc}$ to the empty set and the ordered queue of open nodes $\mathbf{Q}$ to a list including the empty set only (i.e. only the unlabeled root node).

The Main Loop.   Within the loop (line 5) the algorithm gets the node to be processed next, namely the first node node (GETFIRST, line 6) in the list of open nodes **Q** ordered by the function $p_{nodes}()$ and removes node from **Q** (DELETEFIRST, line 7). Note that $p_{nodes}()$ can be directly obtained from $p()$. As mentioned before, for the moment the reader should simply suppose that $p_{nodes}()$ imposes an order on **Q** which effectuates a breadth-first labeling of open nodes in the tree. A definition of $p_{nodes}()$ will be given by Definition 4.9 after a motivation and detailed explanation of $p_{nodes}()$ will have been given in Section 4.6.

Computation of Node Labels.   Then, a label is computed for node in line 8. Nodes are labeled by *valid*, *closed* or a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ by the procedure LABEL (line 18 ff.). This procedure gets as inputs the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, the current node node, the set of already computed minimal conflicts ($\mathbf{C}_{calc}$) and minimal diagnoses ($\mathbf{D}_{calc}$) and the queue **Q** of open nodes, and it returns an updated set of computed minimal conflicts $\mathbf{C}_{calc}$ and a label for node. It works as follows:

A node node is labeled by *closed* iff (a) there is an already computed minimal diagnosis $\mathcal{D}$ in $\mathbf{D}_{calc}$ that is a subset of this node, i.e. $\mathcal{D} \subseteq$ node, which means that node cannot be a minimal diagnosis (non-minimality criterion, lines 19-21) or (b) there is some node nd in the queue of open nodes **Q** such that node = nd which means that one of the two tree branches with an equal set of edge labels can be closed, i.e. removed from **Q** (duplicate criterion, lines 22-24).

If none of these *closed*-criteria is met, the algorithm searches for some $\mathcal{C}$ in $\mathbf{C}_{calc}$, the set of already computed minimal conflict sets, such that $\mathcal{C} \cap$ node $= \emptyset$ and returns the label $\mathcal{C}$ for node (reuse criterion, lines 25-27). This means that the path represented by node cannot be a diagnosis as there is (at least) one minimal conflict set, namely $\mathcal{C}$, that is not hit by node.

If the reuse criterion does not apply, a call to $\mathrm{QX}(\langle \mathcal{K} \setminus$ node$, \mathcal{B}, P, N \rangle_R)$ is made (line 28) in order to check whether there is a not-yet-computed minimal conflict set that is not hit by node. Note that the KB $\mathcal{K} \setminus$ node that is given to QX as part of the argument DPI ensures that only minimal conflict sets $\mathcal{C} \subseteq \mathcal{K} \setminus$node can be computed, i.e. ones that do not share any single formula with node (cf. Section 4.4.1).

Remark 4.3   A minimal conflict set computed by $\mathrm{QX}(\langle \mathcal{K} \setminus$ node$, \mathcal{B}, P, N \rangle_R)$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ indeed since (i) $\mathrm{QX}(\langle \mathcal{K} \setminus$ node$, \mathcal{B}, P, N \rangle_R)$ returning a set $\mathcal{C}$ means that $\mathcal{C}$ is a minimal conflict set w.r.t. $\langle \mathcal{K} \setminus$ node$, \mathcal{B}, P, N \rangle_R$ by Proposition 4.9 and (ii) the "$\Rightarrow$" direction of Corollary 4.1 implies that $\mathcal{C}$ is not valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$ and (iii) the "$\Leftarrow$" direction of Corollary 4.1 lets us conclude that $\mathcal{C}$ is a minimal conflict w.r.t. $\langle X, \mathcal{B}, P, N \rangle_R$ where X is any superset of $\mathcal{C}$, in particular $X := \mathcal{K}$. $\square$

QX may then return (a) 'no conflict', i.e. $\mathcal{K} \setminus$ node is already valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, or (b) a new conflict set $L \neq \emptyset$ such that $L \notin \mathbf{C}_{calc}$. Note that the case of the output $L = \emptyset$ of QX cannot arise since (i) the DPI provided as input to the algorithm is assumed to be admissible, (ii) no other DPI for which QX is called can be non-admissible since admissibility is defined only by the sets $\mathcal{B}, P, N, R$ which remain unmodified throughout the execution of Algorithm 2, and (iii) as per Proposition 4.9, QX returns $\emptyset$ only if the DPI given to it as an argument is non-admissible. Further on, we point out that the conflict set L in case (b) must be a *new* conflict set since the reuse criterion is always checked *before* the call to QX and thus must be negative. That is, each $\mathcal{C} \in \mathbf{C}_{calc}$ is hit by node and L is not hit by node wherefore $L \neq \mathcal{C}$ must hold for all $\mathcal{C} \in \mathbf{C}_{calc}$.

In each of the described cases, the LABEL procedure returns a tuple including the respective label as explained and the set $\mathbf{C}_{calc}$ where $\mathbf{C}_{calc}$ is equal to the input argument $\mathbf{C}_{calc}$ in all cases except for the case where a new minimal conflict set is computed by QX. In this case, the newly computed conflict set is added to $\mathbf{C}_{calc}$ (line 32) before the procedure returns.

Processing of a Node Label.   Back in the main procedure, $\mathbf{C}_{calc}$ is updated (line 9) and then the label L returned by procedure LABEL is processed as follows:

If $L = valid$, then there is no minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that is not hit by (i.e. has an empty intersection with) the current node node. Thus, node is added to the set of calculated minimal diagnoses $\mathbf{D}_{calc}$. Minimality of diagnoses added to $\mathbf{D}_{calc}$ is guaranteed by the pruning rule (lines 19-21) which eliminates non-minimal nodes (paths) and the way the tree is built level by level by the used breadth-first strategy. In case a uniform-cost variant of tree construction is used, certain properties of the function $p()$ need to be postulated to preserve this minimality guarantee. We discuss these properties in Section 4.6.

If, on the other hand, $L = closed$ is the returned label of the procedure LABEL, then there is either a minimal diagnosis in $\mathbf{D}_{calc}$ that is a subset of the current node node or a duplicate of node is already included in $\mathbf{Q}$. Consequently, node must simply be removed from $\mathbf{Q}$ which has already been executed in line 7.

In the third case, if a minimal conflict set L is returned in line 8, then L is a label for node meaning that $|L|$ successor nodes of node need to be added to $\mathbf{Q}$ in sorted order using the function $p_{nodes}()$ (INSERTSORTED, line 15), as will be explained in more detail in Section 4.6.

Recap.   To summarize, in each iteration, the node node that is the first element of the queue $\mathbf{Q}$ is deleted from $\mathbf{Q}$ and,

1. if node is a diagnosis, it is added to the set $\mathbf{D}_{calc}$

2. if there is some diagnosis in $\mathbf{D}_{calc}$ that is a proper subset of node or node is equal to some other node in $\mathbf{Q}$, no action is performed, i.e. the algorithm deletes node without substitution

3. if there is some minimal conflict set that node does not hit, then such a conflict set $\mathcal{C}$ is computed and for each $ax \in \mathcal{C}$ a new node node $\cup \{ax\}$ is added to $\mathbf{Q}$.

We call each node nd that is added to $\mathbf{Q}$ in the latter case a *successor of the node* node.

4.5.2   Correctness of Breadth-First Diagnosis Computation

For the discussion of the output of Algorithm 2 we will exploit the following result saying that Algorithm 2 computes all and only minimal diagnoses, if it executes until the queue of open nodes becomes the empty set.

Proposition 4.11 (Soundness and Completeness of Algorithm 2 using Breadth-First Search)**.** *Let* $\langle \mathcal{K},$ $\mathcal{B}, P, N \rangle_R$ *be an admissible DPI given as input to Algorithm 2. If Algorithm 2 using a breadth-first tree construction strategy terminates due to* $\mathbf{Q} = []$*, then the algorithm returns exactly the set of all minimal diagnoses w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$*.*

Proof. This proposition is a consequence of Proposition 4.10 and the following Lemma 4.13 which witnesses that Algorithm 2 using a breadth-first tree construction strategy produces a pHS-tree as per Definition 4.8. □

Lemma 4.13. *Algorithm 2 with the admissible input DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *using a breadth-first tree construction strategy is a procedure for producing a pHS-tree* T *w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$*.*

Proof. We verify whether all rules given by Definitions 4.7 and 4.8 are satisfied by Algorithm 2.

- Definition 4.7, rule 1: The root node $\emptyset$ which is the only element of the initial list $\mathbf{Q}$ is labeled by the first call to LABEL for node $:= \emptyset$ in line 8. If $valid$ is returned, then QX($\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$) must have returned 'no conflict' which is the case if $\mathcal{K}$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$.

Otherwise, if $valid$ is not returned by LABEL, then some minimal conflict set L w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ must have been returned in line 33. L is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ by Proposition 4.9 and since $\text{QX}(\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$ has not returned 'no conflict' as otherwise $valid$ would have been returned contradicting our assumption and since $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is an admissible DPI by assumption. LABEL cannot have returned earlier in line 21 or line 24, since $\mathbf{D}_{calc}$ is the empty set and $\mathbf{Q}$ the empty list at this time. The former holds since $\mathbf{D}_{calc}$ is only extended in line 11 which cannot ever have been reached before the first call to LABEL has returned. The latter holds as $\mathbf{Q}$ initially contained only $\emptyset$ and as $\emptyset$ was deleted from $\mathbf{Q}$ in line 7 before the call to LABEL was made in line 8.

- Definition 4.7, rule 2: Suppose a node node is labeled by $valid$, then it is added to $\mathbf{D}_{calc}$ in line 11. Since node can only get a label different from $closed$ if it is the only exemplar of this node in $\mathbf{Q}$ due to the duplicate criterion (lines 22-24), it must be the case that node $\notin \mathbf{Q}$ (line 7) after node has been labeled by $valid$. Only nodes that get labeled by a conflict set can have successor nodes added to $\mathbf{Q}$ in line 15. Only nodes in $\mathbf{Q}$ can get a label (cf. lines 6 and 8). For node to be added to $\mathbf{Q}$ at some later point in time there must be a proper subset of node that is still in $\mathbf{Q}$ as each node newly added to $\mathbf{Q}$ is a proper superset of some node in $\mathbf{Q}$ (cf. line 15 which is the only position in the algorithm where nodes are added to $\mathbf{Q}$). This is impossible due to the breadth-first tree construction strategy which implies that all nodes of cardinality $|\text{node}| - 1$ have already been labeled (and thus deleted from $\mathbf{Q}$ in line 7) when node is being labeled. Hence, if node is labeled by $valid$, then it has no successors.

  If node is labeled by some conflict set L, then Algorithm must come to line 15, where a successor node $\cup \{e\}$ is added to $\mathbf{Q}$ for all $e \in L$.

  How node $\text{node}_e := \text{node} \cup \{e\}$ must be labeled is overridden by the rules 3, 4 and 6 of Definition 4.8 (see below).

- Definition 4.8, rule 1: This is true by our assumption about $p()$ and $p_{nodes}()$.

- Definition 4.8, rule 2: This holds since $\text{QX}(\langle \mathcal{K} \setminus \text{node}, \mathcal{B}, P, N \rangle_R)$ computes only minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ (cf. Remark 4.3).

- Definition 4.8, rule 3: All minimal conflict sets that have been used to label nodes so far are stored in $\mathbf{C}_{calc}$. Before a minimal conflict to label node might be computed by a call to QX in line 28, the reuse criterion in lines 25-27 checks whether there is a set $\mathcal{C}$ in $\mathbf{C}_{calc}$ with $\mathcal{C} \cap \text{node}$. If positive, $\mathcal{C}$ is returned as a label for node.

- Definition 4.8, rule 4: This is accomplished by the non-minimality criterion in lines 19-21 which checks for existence of a node already labeled by $valid$ which is a subset of the node to be labeled right now. All nodes labeled by $valid$ are stored in $\mathbf{D}_{calc}$ (cf. lines 10 and 11).

- Definition 4.8, rule 5: If some node node is labeled by $closed$, then no action is performed (cf. line 12). Before each node is labeled in line 8, it is deleted from $\mathbf{Q}$ in line 7. That node cannot be inserted into $\mathbf{Q}$ at some later point in time follows from the argumentation used above to demonstrate that Definition 4.7, rule 2 is met.

- Definition 4.8, rule 6: This is achieved by the duplicate criterion in lines 22-24 where $\mathbf{Q}$ is browsed for some node equal to the one that is to be labeled right now. When some node node is next to be labeled, then all duplicates of node must already be in $\mathbf{Q}$ as reasoned above in the argumentation to show that Definition 4.7, rule 2 is satisfied. Thus, the criterion must search for duplicates in no other collections than $\mathbf{Q}$. Indeed, only one (i.e. the last non-deleted) exemplar of these duplicates of node in $\mathbf{Q}$ can get a label other than $closed$ due to the duplicate criterion which closes duplicates as long as there are any.

We conclude that Algorithm 2 is a procedure for constructing a pHS-tree. □

By Proposition 4.11 and the fact that there is no place in Algorithm 2 where nodes are removed from $\mathbf{D}_{calc}$ (which implies that only *minimal* diagnoses can be added to $\mathbf{D}_{calc}$), the following corollary is obvious.

Corollary 4.4. *Algorithm 2 with the admissible input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ using a breadth-first tree construction strategy stores by $\langle \mathbf{D}_{calc}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ the relevant data of*

- *a pHS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ if Algorithm 2 stops due to $\mathbf{Q} = []$,*

- *a partial pHS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ otherwise.*

If a pHS-tree is computed in breath-first order, minimal diagnoses are generated with increasing cardinality, as the following Corollary 4.5 attests. Consequently, for the generation of all minimum cardinality diagnoses, only the first level of the tree has to be generated, where a node is labeled.

Corollary 4.5. *The following holds for the set $\mathbf{D}$ returned by Algorithm 2 using breadth-first search: If $\mathbf{D}$ contains some diagnosis of cardinality k, then it includes all diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ of cardinality lower than k.*

Proof. By Proposition 4.11, it is a fact that Algorithm 2 computes all and only minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. As these are computed in breadth-first order, the first computed diagnoses must be the minimum cardinality ones. To see this, assume that Algorithm 2 returns $\mathbf{D}$ which includes one non-minimum cardinality diagnosis $\mathcal{D}$ and does not comprise a minimum cardinality diagnosis $\mathcal{D}'$, i.e. $|\mathcal{D}| > |\mathcal{D}'|$. By breadth-first search, nodes are labeled in ascending order of their cardinality. And, if the first node of cardinality k is labeled, no more nodes of cardinality $k-1$ can be in $\mathbf{Q}$ (cf. proof of Lemma 4.13). So, we have that the pHS-tree obtained by further execution of the algorithm until $\mathbf{Q} = []$ can never label $\mathcal{D}'$ since $|\mathcal{D}| > |\mathcal{D}'|$ and $\mathcal{D}$ has already been labeled. Hence, the algorithm would not return $\mathcal{D}'$ in its final output $\mathbf{D}$. Since each minimum cardinality diagnosis is a minimal diagnosis, $\mathcal{D}'$ is a minimal diagnosis. Thus, we have a contradiction to the fact that the algorithm computes all minimal diagnoses. □

Output. The repeat-loop is iterated until the stop criterion (line 16) applies. In case at least $n_{\min}$ minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ exist, there are two cases:

- If the finding of the $n_{\min}$-th minimal diagnosis happens after $t' < t$ time has passed since the start of Algorithm 2, then the algorithm will continue iterating and terminate only if execution time amounts to at least t time or $|\mathbf{D}| = n_{\max}$ at the time line 16 is processed.

- Otherwise, if the detection of the $n_{\min}$-th minimal diagnosis takes place after processing longer than t time, then the algorithm will terminate immediately after having determined the $n_{\min}$-th minimal diagnosis.

In both cases, the output is a set $\mathbf{D}$ of minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ such that $n_{\min} \leq |\mathbf{D}| \leq n_{\max}$ and $\mathbf{D}$ is the set of best minimal diagnoses as per $p()$, in this case the set of minimal diagnoses with minimum cardinality since $p()$ is assumed to be specified as to cause a breadth-first tree construction.

If fewer than $n_{\min}$ minimal diagnoses exist w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, then $\mathbf{Q} = []$ will be the cause for the algorithm to terminate. In this case, the pHS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ has been built up and all minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ are stored in $\mathbf{D}_{calc}$. Thus, the output is the set $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ of all minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Termination. The next proposition shows that Algorithm 2 must yield a set of minimal diagnoses after finite time.

Proposition 4.12. *Algorithm 2 always terminates.*

Proof. This is due to the fact that minimal conflict sets used to label non-leaf nodes are subsets of $\mathcal{K}$ and that nodes in $\mathbf{Q}$ are subsets of $\mathcal{K}$, which is a finite set by Definition 3.1. Moreover, a node in $\mathbf{Q}$ is either deleted without substitution from $\mathbf{Q}$ if *valid* or *closed* (line 7) or deleted (line 7) and replaced by proper supersets of it (INSERTSORTED in line 15). This means that the cardinality of all nodes in $\mathbf{Q}$ is strictly monotonically increasing. Thus each node (path) node is guaranteed to be closed (*valid* or *closed*) when node $= \mathcal{K}$ as in this case node must hit all possible (minimal) conflict sets $\mathcal{C}_i$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ since $\mathcal{C}_i \subseteq \mathcal{K}$ holds by Definition 4.1. So, after finite time the queue $\mathbf{Q}$ definitely becomes the empty list which is a stop criterion (line 16). $\square$

The argumentation so far proves the following

Proposition 4.13. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be an admissible DPI, $t, n_{\min}, n_{\max} \in \mathbb{N}$ and $p : \mathcal{K} \rightarrow (0, 0.5)$ defined in a way that $\mathbf{Q}$ is always ordered first-in-first-out. For these inputs, Algorithm 2 always terminates and returns a set $\mathbf{D}$ of minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ which is*

- *the set of the $|\mathbf{D}|$ minimal diagnoses of minimum cardinality w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ (i.e. the first $|\mathbf{D}|$ elements in $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ if $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ is assumed to be sorted in ascending order by cardinality) such that $n_{\min} \leq |\mathbf{D}| \leq n_{\max}$, if at least $n_{\min}$ minimal diagnoses exist w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, or*

- *the set of all minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, otherwise.*

4.6 Diagnosis Probability Space

The induction of a probability space [Dur10] over diagnoses facilitates incorporation of well-established probability theoretic methods into the process of KB debugging; for example, a Bayesian approach [SFFR12, RSFF13, dKW87] for identifying the *true diagnosis*, i.e. the one which leads to a solution KB with the desired semantics, by repeated measurements (see Part II). Let the true diagnosis be denoted as $\mathcal{D}_t$ in the sequel.

The Probability Space of All Diagnoses. From the point of view of probability theory, a diagnosis can be viewed as an atomic event in a probability space $\langle \Omega, \mathcal{E}, p \rangle$ defined as follows:

- Ω is the sample space consisting of all possible diagnoses w.r.t. a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, i.e. $\Omega = \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$,

- $\mathcal{E}$ is a sigma-algebra on Ω, in our case the powerset 2^Ω of Ω, and

- p is a probability measure assigning a probability to each event in $\mathcal{E}$, i.e. $p : \mathcal{E} \rightarrow [0, 1]$ such that $\sum_{\omega \in \Omega} p(\{\omega\}) = 1$ which means $\sum_{\mathcal{D} \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}} p(\{\mathcal{D}\}) = 1$.

So, $p(\{\mathcal{D}\})$ for $\mathcal{D} \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ can be seen as the probability that $\mathcal{D}$ is the true diagnosis, i.e. the probability of the event $\mathcal{D}_t = \mathcal{D}$ (or $\mathcal{D}_t \in \{\mathcal{D}\}$). Consequently, $p(\{\mathcal{D}\})$ for $\mathcal{D} \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ is the probability distribution of the random variable $\mathcal{D}_t$, i.e. the probability distribution of the true diagnosis. In this vein, the probability of a set $\{\mathcal{D}_i, \ldots, \mathcal{D}_j\} \in \mathcal{E}$ is interpreted as the likeliness of this set to comprise the true diagnosis $\mathcal{D}_t$. That is, $p(\{\mathcal{D}_i, \ldots, \mathcal{D}_j\}) = p(\mathcal{D}_t \in \{\mathcal{D}_i, \ldots, \mathcal{D}_j\}) = p(\mathcal{D}_t = \mathcal{D}_i \vee \cdots \vee \mathcal{D}_t =$

Algorithm 2 HS: Computation of Minimal Diagnoses

Input: an admissible DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, a desired computation timeout t, a desired minimal ($n_{\min}$) and maximal ($n_{\max}$) number of diagnoses to be returned, a function $p : \mathcal{K} \to (0, 0.5)$
Output: a set $\mathbf{D}$ which is
    (a) a set of most probable (according to $p()$) minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ such that $n_{\min} \leq |\mathbf{D}| \leq n_{\max}$, if at least $n_{\min}$ minimal diagnoses exist w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, or
    (b) the set of all minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ otherwise

 1: **procedure** HS($\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R, t, n_{\min}, n_{\max}, p()$)
 2:     $t_{start} \leftarrow$ GETTIME()
 3:     $\mathbf{D}_{calc}, \mathbf{C}_{calc} \leftarrow \emptyset$
 4:     $\mathbf{Q} \leftarrow [\emptyset]$
 5:     **repeat**
 6:         node $\leftarrow$ GETFIRST($\mathbf{Q}$)
 7:         $\mathbf{Q} \leftarrow$ DELETEFIRST($\mathbf{Q}$)
 8:         $\langle L, \mathbf{C} \rangle \leftarrow$ LABEL($\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, node, $\mathbf{C}_{calc}, \mathbf{D}_{calc}, \mathbf{Q}$)
 9:         $\mathbf{C}_{calc} \leftarrow \mathbf{C}$
10:         **if** $L = valid$ **then**
11:             $\mathbf{D}_{calc} \leftarrow \mathbf{D}_{calc} \cup \{$node$\}$
12:         **else if** $L = closed$ **then**   ▷ do nothing
13:         **else**   ▷ L must be a minimal conflict set
14:             **for** $e \in L$ **do**
15:                 $\mathbf{Q} \leftarrow$ INSERTSORTED(node $\cup \{e\}$, $\mathbf{Q}, p_{nodes}()$)
16:     **until** $\mathbf{Q} = [] \lor [|\mathbf{D}_{calc}| \geq n_{\min} \land (|\mathbf{D}_{calc}| = n_{\max} \lor$ GETTIME() $- t_{start} > t)]$
17:     **return** $\mathbf{D}_{calc}$

18: **procedure** LABEL($\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, node, $\mathbf{C}_{calc}, \mathbf{D}_{calc}, \mathbf{Q}$)
19:     **for** nd $\in \mathbf{D}_{calc}$ **do**
20:         **if** node $\supseteq$ nd **then**   ▷ non-minimality
21:             **return** $\langle closed, \mathbf{C}_{calc} \rangle$
22:     **for** nd $\in \mathbf{Q}$ **do**
23:         **if** node $=$ nd **then**   ▷ remove duplicates
24:             **return** $\langle closed, \mathbf{C}_{calc} \rangle$
25:     **for** $\mathcal{C} \in \mathbf{C}_{calc}$ **do**
26:         **if** $\mathcal{C} \cap$ node $= \emptyset$ **then**   ▷ reuse $\mathcal{C}$
27:             **return** $\langle \mathcal{C}, \mathbf{C}_{calc} \rangle$
28:     $L \leftarrow$ QX($\langle \mathcal{K} \setminus$ node$, \mathcal{B}, P, N \rangle_R$)
29:     **if** $L =$ 'no conflict' **then**   ▷ node is a diagnosis
30:         **return** $\langle valid, \mathbf{C}_{calc} \rangle$
31:     **else**   ▷ L is *new* minimal conflict set ($\notin \mathbf{C}_{calc}$)
32:         $\mathbf{C}_{calc} \leftarrow \mathbf{C}_{calc} \cup \{L\}$
33:         **return** $\langle L, \mathbf{C}_{calc} \rangle$

$\mathcal{D}_j) = 0.3$ means that $\mathcal{D}_t$ is an element of $\{\mathcal{D}_i, \ldots, \mathcal{D}_j\}$ with 30% probability. Note that singletons are often written without curly braces, i.e. $p(\{\mathcal{D}_i\})$ is usually written as $p(\mathcal{D}_i)$; we will also do so in the rest of this work.

    The elements of the sample space Ω of a probability space are often called atomic events because they must be *mutually exclusive* (i.e. two atomic events cannot "happen" at the same time as an outcome of the fictive experiment a probability space describes) and *exhaustive* (i.e. for each "execution" of the experiment the probability space describes one atomic event must "happen"). Since the true diagnosis $\mathcal{D}_t$

must be a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and Ω by definition comprises all such diagnoses, exhaustiveness is clearly fulfilled. Mutual exclusiveness is a consequence of the fact that each diagnosis $\mathcal{D}$ gives complete information about the correctness of each formula $ax_k \in \mathcal{K}$. In other words, $\mathcal{D}_t \in \{\mathcal{D}\}$ is a shorthand for the statement that all $ax_i \in \mathcal{D}$ are faulty and all $ax_j \in \mathcal{K} \setminus \mathcal{D}$ are correct. Thus, any two different diagnoses are mutually exclusive events, i.e. $\mathcal{D}_t = \mathcal{D}_i$ implies $\mathcal{D}_t \neq \mathcal{D}_j$ for all $\mathcal{D}_j \in \mathbf{aD}$ such that $\mathcal{D}_i \neq \mathcal{D}_j$.

The probability measure p is completely defined if a probability $p(\mathcal{D})$ for each diagnosis $\mathcal{D} \in \Omega$ is given. Then, by the mutual exclusiveness of events $\mathcal{D}_t \in \{\mathcal{D}_i\}$ and $\mathcal{D}_t \in \{\mathcal{D}_j\}$ for $\mathcal{D}_i \neq \mathcal{D}_j$, the probability

$$p(E) = \sum_{\mathcal{D} \in E} p(\mathcal{D}) \tag{4.1}$$

for each event $E \in \mathcal{E}$.

Restricted Probability Spaces of Diagnoses. In many cases, only a restricted set of diagnoses w.r.t. a DPI is considered relevant for the debugging task. That is, the focus is on locating the true diagnosis among a predefined subset of all diagnoses $\mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. This involves an adaptation of the probability space, in particular of the set Ω. For instance, if not the set of all, but only the set of minimal diagnoses $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ should be considered by a debugging system – as motivated in Section 3.1 – then $\Omega := \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. The other properties $\mathcal{E} = 2^\Omega$ and $\sum_{\omega \in \Omega} p(\{\omega\}) = 1$ remain the same for each restricted probability space, but depend on Ω. Thus, for example, a probability $p(\mathcal{D})$ for $\mathcal{D} \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \subseteq \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ must be generally defined differently, i.e. assigned a higher value, when $\Omega = \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ instead of $\Omega = \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. This is due to the condition that all probabilities of atomic events in Ω must sum up to 1. In practice, because of the computational complexity of diagnosis computation, the used probability space will usually need to be restricted even further in that Ω comprises only a set of "leading diagnoses" which is a subset of all minimal diagnoses w.r.t. a DPI (see Chapter 7).

4.6.1 Construction of a Probability Space

Since a diagnosis constitutes an assumption about the correctness of each formula in the KB, the probability of a diagnosis $\mathcal{D}$ (to be the true diagnosis $\mathcal{D}_t$) can be computed by means of fault probabilities of formulas. In other words, computing the probability of the event $\mathcal{D} = \mathcal{D}_t$ corresponds to computing the probability of the event that exactly all formulas in $\mathcal{D}$ are faulty and all other formulas in the KB are correct.

Estimating Fault Probabilities of Formulas in the KB

Next we discuss various possibilities of how the probability of an $ax \in \mathcal{K}$ might be assessed. To this end, we first make a distinction between situations where some useful empirical data is available or not and then we differentiate between different sorts of such available data and how to take advantage of it.

Empirical Data is Accessible. Let us first reflect on how to utilize different empirical data sources in order to compute formula probabilities. Data can be of the following kinds (enumeration may not be complete):

(a) Regarding formulas: Change logs of formulas in the KB

(b) Regarding the user: Data about common mistakes of the user who has formulated the KB

Ad (a): Prerequisite for the availability of change logs of formulas in the KB is the usage of some KB engineering software with integrated logging or change management. Examples of such KB (ontology) developing environments are Protégé [NSD$^+$00], Web Protégé [TNNM13], SWOOP [KPS$^+$06], OntoEdit [SEA$^+$02] or KAON2.[19] Given a formula $ax \in \mathcal{K}$ and its change log, the fault probability $p(ax)$ of this formula can be estimated by counting the number of modifications accomplished for ax in the change log. The intuition is, the more often ax has been altered, the more uncertain the (set of) author(s) might be about its correctness. This method of probability computation however suffers from a cold-start problem. If a KB is completely newly created, then such information is not available at all. On the other hand, for KBs that are being developed over a long period of time, this method can be assumed to be a rather reliable way of assessing the likeliness of formulas to be faulty.

Ad (b): Clearly, data about common mistakes of a user has to be related to some type of entity that is recurrent and not dependent on a particular KB. Formulas are therefore not suitable and too coarse-grained since one and the same formula will rarely occur in many KBs. More adequate entities to relate a user fault to are predicates (terms) and logical connectives – these usually (re-)appear in many different KBs. In this way, the extrapolation and reusability of collected personal fault information of a user within one KB and between different KBs is granted.

One way of obtaining data about common mistakes of user u on this syntactical level is, for instance, the examination of diagnoses got as a result of past debugging sessions performed on KBs authored by u. Another way is, again, to use the change logs (if available) of formulas in KBs user u has created in the past.

Given such a past diagnosis $\mathcal{D}$, we know that all formulas $ax \in \mathcal{D}$ *that had been written by* u have been confirmed to be faulty by a user. So, these formulas could be analyzed for contained predicates (terms) and logical connectives and the probability of being faulty of those syntactical constructs could be raised relative to those constructs that do not occur in formulas in $\mathcal{D}$. At this, the following assumptions could be made:

- If a formula has been confirmed to be faulty by the user, then the meaning of all predicates (terms) appearing in this formula is not correct (because in the domain that should be modeled the relationship between the predicates (terms) occurring in the formula stated by the formula must not hold). So, all predicates (terms) in ax get more suspicious of being faulty *in general* if $ax \in \mathcal{D}$ for some past solution diagnosis $\mathcal{D}$.

- If a formula including some logical connective is part of some past solution diagnosis, then this type of logical connective gets more suspicious of being faulty *in general*.

When exploiting change logs of formulas *authored by* u, the following assumptions could be made:

- If a formula has been modified, then a user has changed the meaning of all predicates (terms) appearing in this formula. So, all predicates (terms) in ax get more suspicious of being faulty *in general* if ax has been edited at least once. The more often it has been altered, the more suspicious the predicates (terms) get.

- If some logical connective in a formula is modified, i.e. deleted or added, then this type of logical connective gets more suspicious of being faulty *in general*.

The following example should give an intuition of these assumptions:

Example 4.5   Imagine the situation where the author of formula $ax := \forall X \, pet(X) \leftrightarrow animal(X) \land (\exists Y \, hasOwner(X, Y) \land person(Y))$ is known to have only vague knowledge about the predicate *pet*

---

[19]http://kaon2.semanticweb.org/

and to frequently interchange $\wedge$ and $\vee$ when formulating logical formulas. This could be reflected by the assignment of higher fault probability to the predicate pet than to the predicates $animal, hasChild$ and $person$ and by raising the fault probability of $\wedge$ as well as $\vee$ compared to other logical connectives available in the used logic $\mathcal{L}$. Then, formula ax should intuitively have a higher probability of being faulty than, e.g., formula $ax' = \forall X\, animal(X) \rightarrow \neg person(X)$ since ax' does not include any of the "suspicious" terms or connectives as ax does. $\qquad\square$

A probability of 0.25 of some predicate (term) a occurring in $\mathcal{K}$ could then account for the observation made in the logs that, in past debugging sessions (not necessarily related to the current KB $\mathcal{K}$), every fourth formula formulated by user u which includes the term a was modified at least once. Similarly, another term b could be assigned fault probability 0.5 which could reflect that formulas formulated by u including b have been altered twice as often as formulas formulated by u comprising a. Given additionally that a occurred in two formulas formulated by u of past diagnoses whereas b did not occur in any, the probability of a could be increased by some addend or factor to take account of this.

Concerning some logical connective, say $\exists$, the observation that all past diagnosis formulas contained $\exists$ and in 80% of formulas formulated by this user including $\exists$ the $\exists$ connective has been modified at least once, the fault probability of $\exists$ might be assigned rather high. In comparison, the probability of some other connective, say $\neg$, occurring in no diagnosis and having been altered only in 10% of the formulas comprising $\neg$, the probability of the $\neg$ connective might be estimated rather low.

A shortcoming of this approach is again a cold-start problem. If a user is new to conceptualizing knowledge in a structured logical manner or at least in the given logical language $\mathcal{L}$, then no such (personalized) past diagnoses or change logs will be available. So, this issue especially concerns beginners who are usually anyhow more prone to errors than expert-users. On the positive side, utilization of such empirical data can yield to fault information that is very well tailored for the user and that can imply a significant reduction of computation time and user effort necessary for debugging of the KB at hand [SFFR12].

No Empirical Data is Available. If no data of the kinds (a) and (b) discussed above is available to a debugging system, then we have the following possibilities:

(c) Common fault patterns

(d) Subjective self-assessment of a user

(e) Examination of structural complexity of logical formulas

(f) Using no probabilities

Ad (c): A common fault pattern [RDH+04, CRV+09, KPSCG06], also called anti-pattern, refers to a set of formulas that either leads to an inconsistency (logical anti-pattern) or corresponds to a potential modeling error that – alone – does not lead to a inconsistency or incoherency (non-logical anti-pattern), but still might become a source of inconsistency if merged with other formulas (cf. Section 3.2). Although most of these patterns incorporate more than one formula which makes the individual consideration of a formula in terms of fault probability calculation difficult, an idea to incorporate knowledge about anti-patterns to probability estimation of formulas could be to count for each $ax \in \mathcal{K}$ in how many different (logical or non-logical) anti-patterns it occurs. The higher this count, the more likely a formula might be involved in a conflict set and thus in the true diagnosis.

A drawback of this method could be that most of the formulas involved in a KB might not correspond to any formula occurring in an anti-pattern. Thus, one might end up with no probability estimate for most of the formulas in a KB $\mathcal{K}$. Besides that, the information provided by these anti-patterns is not personalized at all and therefore might significantly diverge from the true fault probabilities for a user and lead

to a false bias in the used fault data. This justifies to basically rely on another approach to get a first estimate of a formula's likeliness of being faulty and use this method only to make adaptations to already established probabilities.

Ad (d): The method of a user's self-assessment of own fault probabilities supposes a user to be able to specify fault probabilities of predicates (terms), logical connectives or complete formulas by themselves. Since users not always have a clear picture of own strengths and weaknesses, this variant must be regarded with suspicion. Furthermore, in settings where several persons are involved in the engineering of the KB, a reasonable rating of fault probabilities of terms, connectives or formulas authored by other persons might be difficult or impossible for a user.

Ad (e): Here the idea is to examine "grammatical" (i.e. syntactical) aspects of formulas such as the "nesting depth" of subordinate clauses or the mere "length" of a formula. The underlying assumption can be that higher length and/or deeper nesting means higher complexity and cognitive difficulty in understanding of the formula's semantics – as it does in natural language. For instance, it is reasonable to expect formulas like $ax_1 := \forall X\, a(X) \to (\exists Y\, r_1(X, Y) \land (\forall Z\, r_2(Y, Z) \to b(Z)))$ to tend to be more error-prone and more likely to be faulty than $ax_2 := \forall X\, g(X) \to b(X)$. This intuition is modeled by the maximum nesting depth as well as by the length of ax_1 in comparison to ax_2. Using the analogy to natural language, the maximum nesting depth of a formula could roughly be defined as the maximum number of encapsulated subordinate clauses that cannot be "flattened" occurring in the natural language translation of the formula. For formula ax_1, this would imply a maximum nesting depth of two; for ax_2 it would amount to zero. The reason is that ax_1 stated in natural language would sound "if somebody X is a, then there is somebody Y, who satisfies property r_1 with X and for whom anybody, who satisfies property r_2 with Y is b". In this natural language formulation, there are two subordinate clauses, i.e. the clauses beginning with the word "who"; the first is at nesting depth one and the second at depth two. These subordinate clauses cannot be flattened, i.e. be brought to some lower depth, because the Z is related to the Y which in turn is related to the X. The length of formulas could be defined similarly as in [HPS08] which provides such a definition for DL languages. In this case the length of ax_1 and ax_2 would be four (roughly: four predicates in ax_1) and two (two predicates in ax_2), respectively.

A disadvantage of such a "grammatical" approach gets evident when most of the formulas in a KB are rather "simple", i.e. have a low nesting depth and a short length. In such case this method will give little differentiation between different formulas and should thus be combined with another method of probability estimation in general.

Ad (f): In a situation where all the aforementioned ways of gauging probabilities do not apply or are believed to have a too high risk of introducing a false bias into the debugging system, the solution is to define all formulas to be equally probably faulty. The obvious pro of this is that the system cannot get misled by unreasonable fault probabilities whereas the con is that possibly well-suited probabilistic information cannot be exploited. Moreover, experiments in our previous work [SFFR12] have manifested that fault information of only "average" quality most often leads to a better performance than no fault information. Apart from that, we have suggested a reinforcement learning "plug-in" to a debugger which could successfully mitigate the negative effect of low-quality fault information and in many cases, in spite of the low-quality fault information, even led to lower resource consumption (user, time) than a debugger without this plug-in using good fault information [RSFF13].

Collaborative KB Development.   In a collaborative development scenario involving several authors, provenance information could be additionally leveraged to refine probability estimates (cf. [KPSCG06]). At this point, user skills could come into play; that is, formulas authored by more experienced authors get a lower overall fault probability as opposed to beginners concerning KB engineering or logic skills or

expertise in the modeled domain. This probability adaptation can also affect syntactical elements in that one and the same predicate (term) or logical connective can get a different probability depending on in which formula it occurs and who authored that formula.

Remark 4.4 Of course, these assumptions and methods of obtaining fault probabilities of syntactical elements and formulas are only *some* possible ways of doing so. For example, one might argue that the "authorship" of a formula is somewhat not clearly defined. What if user u_1 has originally written formula ax and then user u_2 alters the formula to become ax'? Who is the author of ax'? u_1, u_2 or both? For whose fault probability computation should the renewed modification of ax' to ax'' count? Questions like this one need to be discussed and maybe evaluations using real data need to be accomplished in order to find a practical answer; or perhaps to find out that completely different approaches turn out to be reasonable. This is a topic of our future work. □

Remark 4.5 By the definition of a DPI (Definition 3.1) stating that the KB $\mathcal{K}$ must be disjoint with the background knowledge $\mathcal{B}$ and the role $\mathcal{B}$ has within a DPI, namely to comprise all formulas that are definitely correct, we postulate that no formula $ax \in \mathcal{K}$ must have a probability of zero. In a situation when this is not the case, a modified DPI must be used where such formulas have been moved from $\mathcal{K}$ to $\mathcal{B}$. □

Computation of Diagnosis Probabilities. In the following, we denote by $\overline{ax}$ ($\overline{\mathcal{K}}$) the set of logical connectives and quantifiers occurring in a formula ax (in the KB $\mathcal{K}$) and by $\widetilde{ax}$ ($\widetilde{\mathcal{K}}$) the signature of ax (of $\mathcal{K}$).

Example 4.6 Considering the DL formula $ax := Pet \equiv Animal \sqcap \exists hasOwner.Person$, we have that $\overline{ax} = \{\equiv, \sqcap, \exists\}$ and $\widetilde{ax} = \{Pet, Animal, hasOwner, Person\}$. □

We now suppose that either a fault probability $p(e) := p(\text{"}e \text{ is faulty"})$ of each element $e \in \overline{\mathcal{K}} \cup \widetilde{\mathcal{K}}$ or the fault probability $p(ax) := p(\text{"}ax \text{ is faulty"})$ of each formula $ax \in \mathcal{K}$ is given. For estimation of these probabilities any (combination) of the methods mentioned above might be employed. In case formula probabilities are given, diagnosis probabilities can be directly computed by Formula 4.3. Otherwise, the following pre-computations must be performed.

The fault probability $p(ax)$ of ax can be calculated as the probability that at least one (occurrence of a) syntactical element in ax is faulty. So, $p(ax)$ is equal to 1 minus the probability that none of the syntactical elements occurring in ax is faulty. Hence, under the assumption of mutual independence of syntactical faults concerning elements $e \in \overline{ax} \cup \widetilde{ax}$,

$$p(ax) = 1 - \prod_{e \in \overline{ax} \cup \widetilde{ax}} (1 - p(e))^{n(e)} \tag{4.2}$$

where $n(e)$ is the number of occurrences of syntactical element e in ax.

If $p(ax)$ for all $ax \in \mathcal{K}$ is known, the fault probability $p(\mathcal{D})$ of any diagnosis $\mathcal{D} \in \Omega \subseteq \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ can be determined as the probability that each formula in $\mathcal{D}$ is faulty whereas each formula in $\mathcal{K} \setminus \mathcal{D}$ is correct, i.e. not faulty. Thence,

$$p(\mathcal{D}) = \prod_{ax_r \in \mathcal{D}} p(ax_r) \prod_{ax_s \in \mathcal{K} \setminus \mathcal{D}} (1 - p(ax_s)) \tag{4.3}$$

Recall that probabilities of all atomic events in a well-defined probability space must sum up to 1. As not every subset of $\mathcal{K}$ is a diagnosis, this is in general not the case. Therefore, diagnosis probabilities

need to be normalized, i.e. each diagnosis probability $p(\mathcal{D})$ must be divided by the sum of all diagnosis probabilities for diagnoses in Ω. That is, the following adjustment is necessary:

$$p(\mathcal{D}) \quad \leftarrow \quad \frac{p(\mathcal{D})}{\sum_{\mathcal{D}_k \in \Omega} p(\mathcal{D}_k)} \tag{4.4}$$

We want to emphasize that the probability measures $p(e)$ of syntactical elements e and $p(ax)$ of formulas ax are not required to satisfy any conditions except for $p(e) \in (0, 1]$ and $p(ax) \in (0, 1]$ for all $e \in \overline{ax} \cup \widetilde{ax}$ and all $ax \in \mathcal{K}$ (see Remark 4.5 why the intervals $(0, 1]$ are open). In particular, no normalization is needed. The reason for this is that "e is faulty" and "ax is faulty" are assumptions about a *single* logical connective and a *single* logical formula, respectively. "$\mathcal{D}$ is the true diagnosis", to the contrary, is an assumption about *each* formula in the KB $\mathcal{K}$. So, the probabilities of two different syntactical elements $e_i \neq e_j$ are computed on the basis of two different probability spaces, namely $\Omega_{e_i} = \{$"e_i is faulty", "e_i is not faulty"$\}$ and $\Omega_{e_j} = \{$"e_j is faulty", "e_j is not faulty"$\}$ which clearly do not depend on each other at all. The same argumentation holds for probabilities of formulas.

More Reliable Probabilities through Observations. As we argued before, the basic fault information from which diagnosis probabilities are deduced might be rather vague. A usual way of dealing with scenarios of that kind, is to regard the initial probabilities as a first (a-priori) estimation and to gather additional information, e.g. by making measurements or observations, and exploit this information to adapt the a-priori estimation in order to obtain a more reliable a-posteriori estimation. The more additional information has been accumulated and incorporated, the more realistic is the resulting updated estimation of probabilities.

A well-known technique enabling computation of a-posteriori probabilities from a-priori probabilities is Bayes' Theorem. Let $p(\mathcal{D})$ be the a-priori probability of some $\mathcal{D} \in \Omega \subseteq \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ and Obs be a new observation. Then, the a-posteriori probability $p(\mathcal{D} \,|\, Obs)$ of $\mathcal{D}$, i.e. the probability that the true diagnosis $\mathcal{D}_t = \mathcal{D}$ taking into account the new information Obs, is computed according to Bayes' Theorem as

$$p(\mathcal{D} \,|\, Obs) = \frac{p(Obs \,|\, \mathcal{D}) \; p(\mathcal{D})}{p(Obs)} \tag{4.5}$$

where $p(Obs)$ is the (a-priori) probability that observation Obs is made and $p(Obs \,|\, \mathcal{D})$ is the (a-priori) probability that the observation Obs is made under the assumption that $\mathcal{D}$ is the true diagnosis, i.e. $\mathcal{D}_t = \mathcal{D}$. That is, the a-priori probability $p(\mathcal{D})$, i.e. the probability that $\mathcal{D}_t = \mathcal{D}$ without any additional knowledge, must be multiplied by $p(Obs \,|\, \mathcal{D})/p(Obs)$ which is often referred to as the *support Obs provides for $\mathcal{D}$*. If the support is greater than 1, then the a-posteriori probability of $\mathcal{D}$ is greater than its a-priori probability, otherwise the a-posteriori probability gets smaller after incorporating the new information Obs. Note that Bayes' Theorem is only applicable to KB debugging if a suitable class of observations can be defined such that $p(Obs)$ and $p(Obs \,|\, \mathcal{D})$ can be computed for observations Obs of this class. As we shall see in Chapter 7, the assignment of test cases to either P or N is one such class of observations. For instance, $t_i \in P$ and $t_j \in N$ for sets of formulas t_i, t_j over $\mathcal{L}$ are two such observations.

4.6.2 Using Probabilities for Diagnosis Computation

If available, formula fault probabilities can be exploited during construction of the pHS-tree (Algorithm 2, Chapter 4) in that most probable instead of minimum cardinality diagnoses are calculated first. To achieve that, breadth-first construction of the tree must be replaced by uniform-cost order of node expansion

by means of the function $p()$ that assigns a fault probability to each formula $ax \in \mathcal{K}$. Thereby, the "probability" $p(\mathsf{nd})$ of a node $\mathsf{nd} = \{ax_s, \ldots, ax_t\}$ in Algorithm 2 is defined through $p(ax)$, $ax \in \mathcal{K}$ as

$$p(\mathsf{nd}) = \prod_{ax_i \in \mathsf{nd}} p(ax_i) \prod_{ax_j \in \mathcal{K} \backslash \mathsf{nd}} (1 - p(ax_j)) \tag{4.6}$$

Notice that this formula extends the definition of Formula 4.3 to arbitrary subsets of $\mathcal{K}$, not only diagnoses. Thus, Formula 4.3 is a special case of Formula 4.6.

First, note that we put "probability" of a node in quotation marks as, to be concise, each node (path) which is not yet a diagnosis, i.e. needs to be further expanded to become one, has probability zero (of being the true diagnosis $\mathcal{D}_t$). For, a probability space is defined on a set of diagnoses and not on a set of arbitrary subsets nd of the KB. However, we misuse the diagnosis probability space in this case to determine the probability of "pseudo-diagnoses" in order to impose an order on the queue of open nodes in the tree. This will guarantee the finding of the most probable diagnoses first, as we shall see below (Proposition 4.17).

Second, note that no normalization, i.e. application of Formula (4.4), is necessary within the scope of the non-interactive Algorithm 2 since the aim here is only the expansion of nodes nd in the order of $p(\mathsf{nd})$ and the return of the most probable identified diagnoses at a certain point in time. For this, the comparison of the probability of one node nd with the probability of another node nd' suffices. Thus, no other calculations using the properties of a probability space are performed by Algorithm 2. We shall recognize in Chapter 9 that this will not hold for the interactive Algorithm 5 where Formula (4.4) is essential.

So, nodes nd are inserted into $\mathbf{Q}$ in a way descending order of node probabilities in $\mathbf{Q}$ is always maintained. Consequently, nodes with highest fault probability are processed first. This is practical since a user will usually be most interested in seeing those possible faults first that have the highest (estimated) probability to be the actual fault they seek.

However, one needs to be careful when using probabilities as weights in order not to lose the property of Algorithm 2 to compute *minimal* diagnoses only. To this end, the formula probabilities $p(ax)$ for all $ax \in \mathcal{K}$ must be adapted as

$$p(ax) \quad \leftarrow \quad c \ p(ax) \tag{4.7}$$

where the factor c is an arbitrary positive real number smaller than 0.5, e.g. $c := 0.49/\max_{\{ax \in \mathcal{K}\}}(p(ax))$. This transformation effects that all probabilities $p(ax)$ become smaller than 50%. In other words, each formula must be more likely to be correct than faulty which in turn means that a minimal diagnosis is more likely than any of its supersets.

Definition 4.9. *Let* $p : \mathcal{K} \to [0, 1]$ *be some function that assigns to each* $ax \in \mathcal{K}$ *some* $p(ax) \in [0, 1]$. *Then, we denote by* $p_{nodes} : 2^{\mathcal{K}} \to [0, 1]$ *the function that assigns to each node* $\mathsf{nd} \subseteq \mathcal{K}$ *some* $p_{nodes}(\mathsf{nd}) \in [0, 1]$ *which is obtained by means of Formula 4.6 and* $p()$.

Lemma 4.14. *Let* $\mathsf{nd}, \mathsf{nd}' \subseteq \mathcal{K}$ *where* $\mathsf{nd} \subset \mathsf{nd}'$ *and* $p : \mathcal{K} \to (0, 0.5)$ *a function which assigns to each* $ax \in \mathcal{K}$ *some probability* $p(ax) \in (0, 0.5)$. *Then* $p_{nodes}(\mathsf{nd}) > p_{nodes}(\mathsf{nd}')$ *holds.*

Proof. According to Formula 4.6 and Definition 4.9 we have that

$$p_{nodes}(\mathsf{nd}) = \prod_{ax_i \in \mathsf{nd}} p(ax_i) \prod_{ax_j \in \mathcal{K} \backslash \mathsf{nd}} (1 - p(ax_j))$$

Then the probability $p_{nodes}(\mathsf{nd}')$ can be computed from $p_{nodes}(\mathsf{nd})$ in that, for each formula ax in $\mathsf{nd}' \backslash \mathsf{nd} \subseteq \mathcal{K} \backslash \mathsf{nd}$, we multiply $p_{nodes}(\mathsf{nd})$ by a factor $f_{ax} := p(ax)/(1 - p(ax))$ because ax "moves" from $\mathcal{K} \backslash \mathsf{nd}$ to nd. However, $f_{ax} < 1$ holds due to $p(ax) < 0.5$ and thus $1 - p(ax) > 0.5$. $\qquad \square$

This result will be a key to proving the completeness, soundness and correctness of Algorithm 2 in the next Section.

The next definition characterizes a (partial) weighted pHS-tree, the type of hitting set tree constructed by Algorithm 2 given any function $p(ax) \in (0, 0.5)$ for all $ax \in \mathcal{K}$ as input which is not necessarily specified in a way a breadth-first tree construction is forced.

Definition 4.10 (Weighted Pruned HS-Tree). *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be an admissible DPI and let $w : \mathcal{K} \to [0, 1]$ be a weight function which assigns a weight to each node $\mathsf{n} \subseteq \mathcal{K}$ with the property that $w(\mathsf{n}_1) > w(\mathsf{n}_2)$ if $\mathsf{n}_1 \subset \mathsf{n}_2$. An edge-labeled and node-labeled tree T is called a* weighted pruned HS-tree (wpHS-tree) *w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $w()$ iff T is the result of constructing an HS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ with due regard to the following rule*

  1. Label open nodes in the HS-tree in order of descending $w()$,

and the rules 2 to 6 as per Definition 4.8.

  T is called a partial weighted pruned HS-tree *w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $w()$ iff T is a weighted pruned HS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $w()$ where not all nodes in T have been labeled yet and non-labeled nodes have no successors.*

Then, we have the following relationship between a (partial) pHS-tree and a (partial) wpHS-tree. An explanation why this holds will be given in Section 4.6.4.

Proposition 4.14. *A (partial) pHS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is a (partial) wpHS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $w()$ where $w()$ is a weight function which, additionally to the property postulated in Definition 4.10, satisfies $w(\mathsf{n}_1) = w(\mathsf{n}_2)$ if $|\mathsf{n}_1| = |\mathsf{n}_2|$.*

  In general, a (partial) wpHS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $w()$ is not a (partial) pHS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Lemma 4.15. *Algorithm 2 is a procedure for producing a wpHS-tree T w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p_{nodes}()$.*

Proof. First, the property $p_{nodes}(\mathsf{n}_1) > p_{nodes}(\mathsf{n}_2)$ if $\mathsf{n}_1 \subset \mathsf{n}_2$ postulated by Definition 4.10 holds by Lemma 4.14 and the fact that the function p given as input to Algorithm 2 satisfies $p(ax) \in (0, 0.5)$ for all $ax \in \mathcal{K}$. Moreover, the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ provided as an input to Algorithm 2 is admissible, as postulated by Definition 4.10.

The compliance with rule 1 of Definition 4.7 as well as with rules 2 to 6 of Definition 4.8 is a simple consequence of Lemma 4.13. In the following we prove that rule 2 of Definition 4.7 and rule 1 of Definition 4.10 are satisfied.

- Definition 4.7, rule 2: Suppose a node nd is labeled by $valid$. Then it is added to $\mathbf{D}_{calc}$ in line 11. Since nd can only get a label different from $closed$ if it is the only exemplar of this node in $\mathbf{Q}$ due to the duplicate criterion (lines 22-24), it must be the case that $\mathsf{nd} \notin \mathbf{Q}$ (line 7) after nd has been labeled by $valid$. Only nodes that get labeled by a conflict set can have successor nodes added to $\mathbf{Q}$ in line 15. Only nodes in $\mathbf{Q}$ can get a label (cf. lines 6 and 8). For nd to be added to $\mathbf{Q}$ at some later point in time there must be a proper subset of nd that is still in $\mathbf{Q}$ as each node newly added to $\mathbf{Q}$ is a proper superset of some node in $\mathbf{Q}$ (cf. line 15 which is the only position in the algorithm where nodes are added to $\mathbf{Q}$). This is impossible since $\mathbf{Q}$ is ordered descending by $p_{nodes}()$. Hence, each proper subset of nd must have been ranked before nd in $\mathbf{Q}$ and thus must have already been labeled because nd is already labeled by assumption. Hence, if nd is labeled by $valid$, then it has no successors.

- Definition 4.10, rule 1: That nodes are processed and labeled in order of descending $p_{nodes}()$ follows from the fact that new nodes are inserted into $\mathbf{Q}$ only in a way that the order of $\mathbf{Q}$ by descending $p_{nodes}()$ is maintained (INSERTSORTED in line 15) and by the fact that always the first element of $\mathbf{Q}$ is selected to be labeled next (GETFIRST in line 6).

This completes the proof. □

Let the relevant data of a wpHS-tree be defined as for a pHS-tree (cf. Remark 4.2). By the correctness of Lemma 4.15, we have:

Corollary 4.6. *Algorithm 2 stores by* $\langle \mathbf{D}_{calc}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ *the relevant data of*

- *a wpHS-tree w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *and* $p_{nodes}()$ *if Algorithm 2 stops due to* $\mathbf{Q} = []$, *and*

- *a partial wpHS-tree w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *and* $p_{nodes}()$ *otherwise.*

4.6.3 Correctness of Weighted Diagnosis Computation

First, we show the completeness of Algorithm 2 regarding minimal diagnoses, i.e. that it computes *all* minimal diagnoses w.r.t. the DPI it is given as input.

Lemma 4.16. *Only diagnoses w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *can be added to* $\mathbf{D}_{calc}$ *by Algorithm 2.*

Proof. A node nd can be added to $\mathbf{D}_{calc}$ only in line 11. To reach this line, LABEL must have returned *valid* for nd. For this to hold, $\text{QX}(\langle \mathcal{K} \setminus \text{nd}, \mathcal{B}, P, N \rangle_R)$ must have returned 'no conflict' which implies that nd is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ by Propositions 4.9 and 3.2. □

Lemma 4.17. *Let T denote a (partial) wpHS-tree produced by Algorithm 2. Further, let $\mathbf{Q}$ be the queue of open nodes in T maintained by Algorithm 2 and let nd be some node which occurs only once in $\mathbf{Q}$ and which is a proper subset of some minimal diagnosis w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. *Then:*

(1) The nodes $\emptyset = \text{nd}_1, \ldots, \text{nd}_k$ along any path from the root node $\emptyset$ to nd_k in T satisfy $\text{nd}_i \subset \text{nd}_{i+1}$ and $|\text{nd}_i| + 1 = |\text{nd}_{i+1}|$ and $\text{nd}_i \subseteq \mathcal{K}$ for $1 \leq i \leq k$.

(2) If the LABEL function is called for nd, then it yields some minimal conflict set $\mathcal{C}$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ with $\text{nd} \cap \mathcal{C} = \emptyset$.

Proof. (1): In the representation used by Algorithm 2, a node nd in the (partial) wpHS-tree T produced by Algorithm 2 is defined as the set of all edge labels on the path from the root node to nd (see Remark 4.2) and the successor of a node is defined as a node added to $\mathbf{Q}$ after nd has been labeled by a minimal conflict set. After the LABEL function for node nd has returned some minimal conflict set L as a label for nd, Algorithm 2 goes to line 15 since $L \neq closed$ and $L \neq valid$ and adds an element $\text{nd} \cup \{e\}$ to $\mathbf{Q}$ for each $e \in L$. Therefore, it holds that $|\text{nd} \cup \{e\}| = |\text{nd}| + 1$ for each successor of nd. Hence, $\text{nd}_i \subset \text{nd}_{i+1}$ and $|\text{nd}_i| + 1 = |\text{nd}_{i+1}|$ holds for any path of nodes $\emptyset = \text{nd}_1, \ldots, \text{nd}_k$ in T starting from the root node.

The argumentation why each node must be a subset of $\mathcal{K}$ is as follows: Suppose $\text{node} \cup \{e\}$ is added to $\mathbf{Q}$ in line 15 which is the only place in Algorithm 2 where nodes are added to $\mathbf{Q}$. So, LABEL must have returned neither *valid* nor *closed* for node. Hence, node cannot be a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ as otherwise LABEL with argument node must have returned *valid* in line 30. Due to the fact that $\text{node} = \mathcal{K}$ is definitely a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ as it must hit all minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ which must all be subsets of $\mathcal{K}$ (Definition 4.1), $\text{node} \subset \mathcal{K}$ must hold.

(2): Suppose the LABEL function is called for a node $\text{nd} \in \mathbf{Q}$ where $\text{nd} \subset \mathcal{D}$ for some minimal diagnosis $\mathcal{D}$.

First, there cannot be any $\text{nd}' \in \mathbf{D}_{calc}$ with $\text{nd}' \subseteq \text{nd}$ since $\mathbf{D}_{calc}$ includes only diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $\text{nd} \subset \mathcal{D}$ wherefore there would be a diagnosis $\text{nd}' \subset \mathcal{D}$, contradiction. Due to the fact that nd is present only once in $\mathbf{Q}$, there cannot be some $\text{nd}' = \text{nd}$ in $\mathbf{Q}$. Thus, *closed* cannot be returned for nd by LABEL.

By the facts that a diagnosis must hit all minimal conflict sets (Proposition 4.6) and that nd is a proper subset of a diagnosis, either the criterion checked in line 26 must be true or $\text{QX}(\langle \mathcal{K} \setminus \text{nd}, \mathcal{B}, P, N \rangle_R)$

must return a minimal conflict set L, i.e. $L \neq$ 'no conflict'. In both cases, a minimal conflict set is returned by LABEL.

There are no other labels that can be returned by LABEL.   □

Lemma 4.18. *Each minimal diagnosis w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *occurs as a node in* $\mathbf{Q}$ *during the execution of Algorithm 2, if the execution stops due to* $\mathbf{Q} = []$.

Proof. For Algorithm 2 it holds that

(i) if nd is the last exemplar of some node in $\mathbf{Q}$ which is a proper subset of some minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and the LABEL function is called for nd, then it yields some minimal conflict set $\mathcal{C}$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ with nd $\cap \, \mathcal{C} = \emptyset$ by Lemma 4.17 and

(ii) each node nd that has been labeled by some minimal conflict set $\mathcal{C}$ is deleted from $\mathbf{Q}$ (line 7) whereupon one successor node nd$_{ax}$ = nd$\cup\{ax\}$ for each element $ax \in \mathcal{C}$ is added to $\mathbf{Q}$ (INSERTSORTED in line 23) and

(iii) each minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is a superset of $\emptyset$ and a subset of $\mathcal{K}$ (Definition 3.5) which includes one element of each minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and includes only elements of minimal conflict sets (Proposition 4.6).

Let $\mathcal{D}$ be some minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then, there is a path of nodes from the root node $\emptyset$ to $\mathcal{D}$ in the pHS-tree produced by Algorithm 2, if the execution stops due to $\mathbf{Q} = []$.

This holds by the following argumentation: If $\mathcal{D} = \emptyset$, then the path is $\langle \emptyset \rangle$. Now, suppose $\mathcal{D} \supset \emptyset$. Since $\mathcal{D}$ is a minimal diagnosis wherefore no other diagnosis can be equal to $\emptyset$, the root node $\mathsf{n}_0 := \emptyset$ of the constructed tree must be labeled by some minimal conflict set $\mathcal{C}_1$. Then, by (iii), there must be some $ax_1 \in \mathcal{C}_1$ that is an element of $\mathcal{D}$. So, we define $\mathsf{n}_1 := \{ax_1\}$. If $\mathsf{n}_1 = \mathcal{D}$, then the path is $\langle \emptyset, \mathsf{n}_1 \rangle$. Otherwise, due to $\mathcal{D} \supset \mathsf{n}_1$ and (i), node n_1 in the pHS-tree must be labeled by some minimal conflict set $\mathcal{C}_2$. Then, by (iii), there must be some $ax_2 \in \mathcal{C}_2$ that is an element of $\mathcal{D}$. So, we define $\mathsf{n}_2 := \mathsf{n}_1 \cup \{ax_2\}$. If $\mathsf{n}_2 = \mathcal{D}$, then the path is $\langle \emptyset, \mathsf{n}_1, \mathsf{n}_2 \rangle$. Otherwise, due to $\mathcal{D} \supset \mathsf{n}_2$ and (i), node n_2 in the pHS-tree must be labeled by some minimal conflict set $\mathcal{C}_3$. This reasoning can be continued until $\mathsf{n}_k = \mathcal{D}$ for some k. By (iii), $\mathcal{D} \subseteq \mathcal{K}$ holds wherefore such k must exist.

Algorithm 2 cannot stop executing before n_k has been in $\mathbf{Q}$ since each node n_i labeled by a minimal conflict set $\mathcal{C}_{i+1}$ involves the addition of $|\mathcal{C}_{i+1}|$ successor nodes to $\mathbf{Q}$ by (ii). In particular, the successor node $\mathsf{n}_i \cup \{ax_{i+1}\}$ must be added to $\mathbf{Q}$. As the execution stops due to $\mathbf{Q} = []$, all nodes n_i for $i \leq k$ must be labeled before termination. Thus, $\mathcal{D}$ must be in $\mathbf{Q}$ sometime.   □

Proposition 4.15 (Completeness of Algorithm 2). *If Algorithm 2 terminates due to* $\mathbf{Q} = []$, *then the algorithm returns a set* $\mathbf{D}$ *including all minimal diagnoses w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Proof. Assume some minimal diagnosis $\mathcal{D}$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ where $\mathcal{D} \notin \mathbf{D}$ after Algorithm 2 has returned due to $\mathbf{Q} = []$. First, each minimal diagnosis will occur in $\mathbf{Q}$ throughout the execution of Algorithm 2 because it executes until $\mathbf{Q} = []$ wherefore Lemma 4.18 applies. Any node nd in $\mathbf{Q}$ can only be deleted from $\mathbf{Q}$ if LABEL is called with the argument node nd (lines 7 and 8). There is no other point in Algorithm 2 where elements are removed from $\mathbf{Q}$. Since at the end $\mathbf{Q} = []$, each minimal diagnosis, in particular $\mathcal{D}$, must be labeled.

Suppose $\mathcal{D}$ is the last exemplar of possibly multiple duplicates of it in $\mathbf{Q}$. Then, the LABEL function cannot return *closed* for $\mathcal{D}$. This holds, on the one hand, because the duplicate criterion (lines 22-24) only removes possible duplicate nodes from $\mathbf{Q}$, but never the last exemplar of a node in $\mathbf{Q}$. On the other hand, $\mathcal{D}$ can never be closed due to the non-minimality criterion (lines 19-21) as $\mathbf{D}_{calc}$ can only include diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ by Proposition 4.16. Thus, due to the minimality of $\mathcal{D}$, $\mathbf{D}_{calc}$ cannot

comprise any diagnosis $\mathcal{D}'$ with $\mathcal{D}' \subseteq \mathcal{D}$, except for some $\mathcal{D}'$ which is equal to $\mathcal{D}$. This would however be a contradiction to the assumption that $\mathcal{D} \notin \mathbf{D}$.

The reuse criterion (lines 25-27) cannot apply for $\mathcal{D}$ either since a minimal diagnosis is a hitting set of all minimal conflict sets (Proposition 4.6) wherefore there cannot be a minimal conflict set in $\mathbf{C}_{calc}$ which has an empty intersection with $\mathcal{D}$. So, the algorithm will come to line 28 where $\mathrm{QX}(\langle \mathcal{K} \setminus \mathcal{D}, \mathcal{B}, P, N \rangle_R)$ will return 'no conflict' (Propositions 4.9 and 3.2). Therefore, $\mathcal{D}$ will be labeled by *valid* and will be added to $\mathbf{D}_{calc}$ in line 11. $\qquad\square$

Next, we show the soundness of Algorithm 2 w.r.t. minimal diagnoses, i.e. that it computes *only* minimal diagnoses w.r.t. the DPI it is given as input.

Proposition 4.16 (Soundness of Algorithm 2). *If an element $\mathcal{D}$ is added to the set $\mathbf{D}_{calc}$ during the execution of Algorithm 2, $\mathcal{D}$ is a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Proof. Assume that some element nd is added to $\mathbf{D}_{calc}$ which is not a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. This immediately yields a contradiction due to Lemma 4.16.

Assume now that some element nd is added to $\mathbf{D}_{calc}$ which is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, but not a minimal one. Now, since nd is a non-minimal diagnosis, there is some $\mathcal{D} \subset$ nd which is a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Then, there are three cases to distinguish: (a) $\mathcal{D}$ is in $\mathbf{Q}$ and (b) $\mathcal{D}$ is in $\mathbf{D}_{calc}$ and (c) $\mathcal{D}$ is neither in $\mathbf{Q}$ nor in $\mathbf{D}_{calc}$, i.e. the node $\mathcal{D}$ has not yet been generated.

Note that these are all possible cases as $\mathcal{D}$ is a *minimal* diagnosis by assumption. So, $\mathcal{D}$ cannot have been ruled out, i.e. labeled by *closed*, by the non-minimality criterion (lines 19-21) before since only diagnoses can be added to $\mathbf{D}_{calc}$ as argued in the first paragraph of this proof and there cannot be a diagnosis $\mathcal{D}' \in \mathbf{D}_{calc}$ such that $\mathcal{D}' \subset \mathcal{D}$. The case $\mathcal{D}' = \mathcal{D}$ is already considered by case (b). The duplicate criterion (lines 22-24) does not need to be taken into account since it deletes duplicate nodes only.

(a): To be added to $\mathbf{D}_{calc}$, nd must have been the first element of the queue $\mathbf{Q}$ by GETFIRST in line 6. Since $\mathcal{D} \in \mathbf{Q}$ by assumption and since $\mathbf{Q}$ is sorted in descending order of node probability (INSERTSORTED in line 15), we conclude that $p_{nodes}(\mathcal{D}) \leq p_{nodes}(\mathsf{nd})$. However, as $p_{nodes}(X)$ for a node $X \subseteq \mathcal{K}$ is defined by means of $p(ax)$ where $p(ax) \in (0, 0.5)$ for all $ax \in \mathcal{K}$ as per Formula 4.6 (Definition 4.9), Lemma 4.14 applies and establishes the truth of $p_{nodes}(S_1) > p_{nodes}(S_2)$ if $S_1 \subset S_2$ for $S_1, S_2 \subseteq \mathcal{K}$. By $\mathcal{D} \subset$ nd, this implies $p_{nodes}(\mathcal{D}) > p_{nodes}(\mathsf{nd})$, contradiction.

(b): Assuming case (b), we can derive a contradiction as follows. By the fact that nd is added to $\mathbf{D}_{calc}$, it must hold that the LABEL procedure called for nd in line 8 returned *valid* as part of its output in line 30. However, as $\mathcal{D} \subset$ nd is already an element of $\mathbf{D}_{calc}$ by assumption, the LABEL procedure must have already returned in line 21 wherefore it cannot have reached line 30, contradiction.

(c): Suppose that $\mathcal{D}$ has not yet been generated as a node in $\mathbf{Q}$. By Lemma 4.17, the nodes $\emptyset = \mathsf{nd}_1, \ldots, \mathsf{nd}_k$ along a path from the root node in the pHS-Tree produced by Algorithm 2 satisfy $\mathsf{nd}_i \subset \mathsf{nd}_{i+1}$ and $|\mathsf{nd}_i| + 1 = |\mathsf{nd}_{i+1}|$. So, by Lemma 4.14, the node probabilities along any path from the root node are strictly monotonically decreasing. Since $p_{nodes}(\mathcal{D}) > p_{nodes}(\mathsf{nd})$ holds by the same argumentation as in (a), we have that all nodes on the path from the root node to $\mathcal{D}$ have a higher probability than nd. As $\mathbf{Q}$ is sorted in descending order of node probability and in each iteration the first element in $\mathbf{Q}$ is processed as explained in (a), we infer that $\mathcal{D}$ must have already been generated at the time nd is processed, contradiction. $\qquad\square$

Next, we argue that Algorithm 2 computes minimal diagnoses in descending order of diagnosis probability according to the parameter $p()$ given as input to the algorithm.

Corollary 4.7. *Let the probability $p(\mathcal{D})$ of a diagnosis $\mathcal{D}$ in Algorithm 2 be computed from the given function $p(ax)$, $ax \in \mathcal{K}$ as per Formula 4.3.*

1. *At any point in time during the execution of Algorithm 2, $\mathbf{D}_{calc}$ comprises the $|\mathbf{D}_{calc}|$ most probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

2. *If Algorithm 2 returns a set $\mathbf{D}$ of cardinality n, then $\mathbf{D}$ is the set of the n most-probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Proof. (1): By Propositions 4.15 and 4.16, it is a fact that Algorithm 2 computes all and only minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. What must still be shown is that minimal diagnoses are added to $\mathbf{D}_{calc}$ in descending order of their probability $p()$ as per Formula 4.3. The probability $p(\mathcal{D})$ of some diagnosis $\mathcal{D}$ is equal to $p_{nodes}(\mathcal{D})$ since a each diagnosis is a node and Formula 4.3 is a special case of Formula 4.6 by which the probability $p_{nodes}(\mathsf{nd})$ of a node nd is calculated.

Let us denote by $\mathcal{D}_{pmax}$ the minimal diagnosis with maximum probability that has not yet been added to $\mathbf{D}_{calc}$ and by $\mathcal{D}_{\neg pmax}$ an arbitrary minimal diagnosis with non-maximal probability, that is $p_{nodes}(\mathcal{D}_{\neg pmax}) < p_{nodes}(\mathcal{D}_{pmax})$. So, we need to demonstrate that each node $\mathsf{nd} \subset \mathcal{D}_{pmax}$ on a path from the root node to node $\mathcal{D}_{pmax}$ is processed before $\mathcal{D}_{\neg pmax}$ is treated. By Lemma 4.17, a path from the root node in the pHS-Tree produced by Algorithm 2 is a set of nodes $\emptyset = \mathsf{nd}_1, \ldots, \mathsf{nd}_k$ where $\mathsf{nd}_i \subset \mathsf{nd}_{i+1}$ and $|\mathsf{nd}_i| + 1 = |\mathsf{nd}_{i+1}|$. Further recall that the probability $p_{nodes}(X)$ of a node $X \subseteq \mathcal{K}$ in Algorithm 2 is defined as per Formula 4.6. So, by Lemma 4.14, the node probabilities along any path from the root node are strictly monotonically decreasing. Hence, each node nd on a path from the root node to $\mathcal{D}_{pmax}$ has a probability $p_{nodes}(\mathsf{nd}) > p_{nodes}(\mathcal{D}_{pmax}) > p_{nodes}(\mathcal{D}_{\neg pmax})$. By the insertion of new nodes into $\mathbf{Q}$ (INSERTSORTED in line 15) in a way descending order of $\mathbf{Q}$ as per $p_{nodes}()$ is always maintained, and by the selection of the first element of $\mathbf{Q}$ (GETFIRST in line 6) as next node to be processed, each node nd on a path to $\mathcal{D}_{pmax}$ must be processed before $\mathcal{D}_{\neg pmax}$ is processed. Consequently, minimal diagnoses are added to $\mathbf{D}_{calc}$ in descending order of their probability $p()$ as per Formula 4.3.

(2): This proposition follows directly from (1). $\hfill\square$

Proposition 4.17. *Algorithm 2 always terminates and returns a set $\mathbf{D}$ of minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ which is*

- *the set of the $|\mathbf{D}|$ most probable (w.r.t. $p()$ and Formula 4.3) minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ such that $n_{\min} \leq |\mathbf{D}| \leq n_{\max}$, if at least $n_{\min}$ minimal diagnoses exist w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, or*

- *the set of all minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, otherwise.*

Proof. The proposition is a direct consequence of Propositions 4.12, 4.15 and 4.16 and Corollary 4.7. $\hfill\square$

4.6.4   Using Probabilities to Compute Minimum Cardinality Diagnoses

The function $p : \mathcal{K} \to (0, 0.5)$ can be defined in a way that minimum cardinality instead of maximum probability diagnoses are identified first. To this end, $p()$ is specified as a fixpoint function that maps each formula $ax \in \mathcal{K}$ to *one and the same* constant value $p(ax) := c$ where c is an arbitrary real number such that $0 < c < 0.5$, e.g. $c := 0.3$. That in this setting diagnoses are found in order of ascending cardinality is a simple consequence of Corollary 4.7.

Example 4.7   Let us now study how such formula and diagnosis probabilities would be constructed for the example DPI depicted by Table 15.3. Let us suppose that the KB $\mathcal{K}$ in the DPI was formulated by a single user u for whom the personal fault probabilities of syntactical elements $\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}$ given by the first row of Table 4.4 have been extracted from log data of the KB editing software applied by u. Then, the resulting probabilities of formulas $ax \in \mathcal{K}$ as per Formula 4.2 are as presented in the rightmost column of Table 4.4. The entries in the table from the second to the last but two column display the number of occurrences of the syntactical element given by the column label in the formula given by the row label. These values are required to compute the formula probabilities listed in the last but one column

fault prob.	0.25	0.01	0.03	0.05	0.4	0.1	0.6	0.6	0.01	0.25	0.05	0.05		
	terms $\widetilde{\mathcal{K}}$								logical conn. $\overline{\mathcal{K}}$				after Eq. 4.2	after Eq. 4.7
$ax \in \mathcal{K}$	A	B	E	F	G	X	Y	Z	$\rightarrow$	$\neg$	$\wedge$	$\vee$	$p(ax)$	$p(ax)$
ax_1	1		1						1				0.28	0.14
ax_2			1	1		1	1	1	1		2	1	0.89	0.43
ax_3		1	1						1				0.07	0.03
ax_4		1			1				1				0.12	0.06
ax_5	1						1		1	1			0.78	0.38
ax_6		1						1	1				0.61	0.30
ax_7				1			1		1				0.76	0.37

Table 4.4: Computing fault probabilities of formulas in $\mathcal{K}$ given fault probabilities of syntactical elements $e \in \widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}$ for the DPI given by Table 15.3.

as per Formula 4.2. The final probabilities that can "safely" be incorporated into Algorithm 2 under a guarantee that only minimal diagnoses will be output are shown in the last column. These result from an application of Formula 4.7 to the probabilities given in the last but one column with an adaptation parameter $c := 0.49$.

Notice that, for example, $p(ax_5)$ is rather high since the predicates A and Y as well as the connective $\neg$ occurring in ax_5 have a comparably high fault probability in relation to syntactical elements appearing in other formulas. Formula ax_3, on the other hand, comprises only two predicates which should be well-understood by u and no connectives except for $\rightarrow$ which is not problematic for u either. Therefore, its fault probability is rather low. □

4.7 Non-Interactive Knowledge Base Debugging Algorithm

Algorithm 3 describes the procedure for non-interactive debugging of KBs. The algorithm requires as input all the parameters that are required by Algorithm 2 and an additional parameter $auto \in \{true, false\}$ indicating either automatic ($true$) or manual ($false$) mode. If $auto = false$, Algorithm 3 calls HS (Algorithm 2) with the parameters as provided. The set of minimal diagnoses $\mathbf{D}$ returned by HS is then presented to the user who can select a diagnosis manually after inspecting the diagnoses in $\mathbf{D}$. Alternatively, in case of $auto = true$, the system calls HS with the parameters as provided, but with $n_{\min} = n_{\max} = 1$. Hence, only the most probable minimal diagnosis is computed by HS and returned as an output of Algorithm 3 to the user.

If a user wants the algorithm to output the set of all minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, then the parameter setting $auto = false$ and $n_{\min} = \infty$ must be chosen. If, on the other hand, a fixed number n of leading diagnoses should be computed (as long as there are at least n minimal diagnoses for the DPI), then $n_{\min} := n =: n_{\max}$ are the correct parameter settings. Note that in both cases the specification of t has no effect.

Of course, the user can also apply Algorithm 3 several times with varying parameters $t, n_{\min}, n_{\max}$ and $p()$. Or they can specify a test case, i.e. add a set of formulas X either to P (if each $ax \in X$ should be entailed by the correct KB) or to N (if *the conjunction of* all formulas in X must not be implied by the correct KB), and rerun the algorithm with this modified DPI.

Algorithm 3 Non-Interactive KB Debugging
___

Input: a tuple $\langle\langle\mathcal{K}, \mathcal{B}, P, N\rangle_R, t, n_{\min}, n_{\max}, p(), auto\rangle$ consisting of

- an admissible DPI $\langle\mathcal{K}, \mathcal{B}, P, N\rangle_R$,

- some computation timeout t,

- a desired minimal ($n_{\min}$) and maximal ($n_{\max}$) number of diagnoses to be returned,

- a function $p : \mathcal{K} \to (0, 0.5)$ and

- a boolean parameter $auto \in \{true, false\}$.

Output: a set $\mathbf{D}$ which is

   (a) the set of the $|\mathbf{D}|$ most probable minimal diagnoses w.r.t. $\langle\mathcal{K}, \mathcal{B}, P, N\rangle_R$ such that $n_{\min} \leq |\mathbf{D}| \leq n_{\max}$, if at least $n_{\min}$ minimal diagnoses exist w.r.t. $\langle\mathcal{K}, \mathcal{B}, P, N\rangle_R$, or

   (b) the set of all minimal diagnoses w.r.t. $\langle\mathcal{K}, \mathcal{B}, P, N\rangle_R$ otherwise

   where "most-probable" refers to the probability measure $p_{nodes}()$ (cf. Definition 4.9) obtained from the given function $p()$.

1: **if** $auto = true$ **then**
2:     $\mathbf{D} \leftarrow \text{HS}(\langle\mathcal{K}, \mathcal{B}, P, N\rangle_R, t, 1, 1, p())$ ▷ see Algorithm 2
3: **else**
4:     $\mathbf{D} \leftarrow \text{HS}(\langle\mathcal{K}, \mathcal{B}, P, N\rangle_R, t, n_{\min}, n_{\max}, p())$ ▷ see Algorithm 2
5: **return** $\mathbf{D}$

Anyway, the user must either find the correct diagnosis (if it is an element of the output set $\mathbf{D}$ at all) by hand or be convinced that the returned minimum cardinality or respectively maximum probability diagnosis is indeed the one that yields a solution KB with the intended semantics. Moreover, when formulating test cases by hand, a user can be assumed to be as likely to specify something contradictory or faulty as during creation of the KB itself.

Unsurprisingly, application of Algorithm 3 will often lead to unsatisfying solution ontologies. Remedy for this is provided by Interactive KB Debugging which on the one hand requires higher effort of one (or several) user(s), but on the other hand ensures a high quality solution in terms of its semantics to the problem of Parsimonious KB Debugging (Problem Definition 3.2).

Example 4.8 Assume a user wants to find a maximal solution KB for the example DPI $\langle\mathcal{K}, \mathcal{B}, P, N\rangle_R$ provided by Table 15.3 and that no data giving information about fault probabilities of syntactical constructs or formulas in $\mathcal{K}$ is available. Therefore, let $p(ax) := c$ for some fixed $c \in (0, 0.5)$ (see Section 4.6.2 for an explanation of this choice of c). The non-interactive KB debugging algorithm presented by Algorithm 3 called with $\langle\mathcal{K}, \mathcal{B}, P, N\rangle_R$, the function $p()$, $n_{\min} = \infty$ and $auto = false$ as inputs results in the hitting set tree given by the upper picture in Figure 4.2. By $n_{\min} = \infty$ and $auto = false$, the user signalizes that *inspection of all* minimal diagnoses w.r.t. the input DPI is desired. Hence, the (complete) breadth-first pHS-tree as per Algorithm 2 is constructed. So, the output is the set of all minimal diagnoses $\mathbf{mD}_{\langle\mathcal{K}, \mathcal{B}, P, N\rangle_R} = \{[1], [2], [5, 7]\}$.

In the shown hitting set tree, minimal diagnoses are indicated by nodes labeled by $\checkmark_{(\mathcal{D})}$ where $\mathcal{D}$ is a name given to this diagnosis. A node closed due to non-minimality is denoted by $\times_{(\supset\mathcal{D})}$ where $\mathcal{D}$ is some minimal diagnosis that is a subset of the set of edge labels along the path leading from the root node to this node. The label $\mathcal{C}^C$ means that the minimal conflict set $\mathcal{C}$ has been freshly computed by a call to QX. The label $\mathcal{C}^R$, on the other hand, means that the minimal conflict set $\mathcal{C}$ has been reused from the set of already computed minimal conflict sets. In this example, both minimal conflict sets are computed by QX and no conflict sets are reused. The order of node labeling is indicated by the numbers ⓘ starting from 1. Open nodes, i.e. generated nodes that have not yet been labeled, are indicated by a question mark.
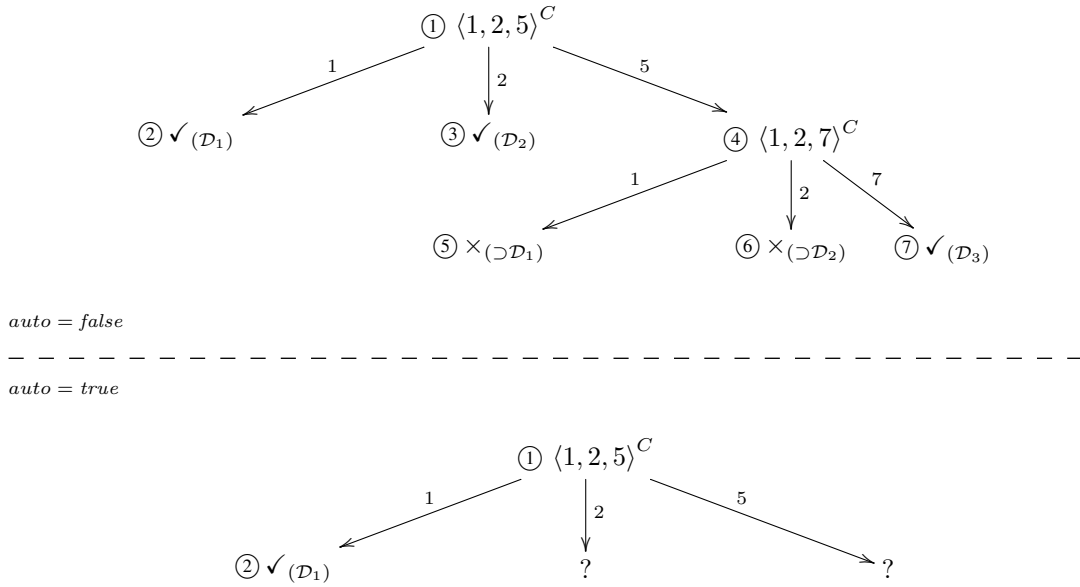
Figure 4.2: Non-interactive KB debugging process without any given fault information applied to the DPI given by Table 15.3 with settings $auto = false$ and $n_{\min} = \infty$ (above) and $auto = true$ (below).

In case $auto = true$ was given as an input to the algorithm instead, the partial pHS-tree depicted by the lower picture in Figure 4.2 would be constructed and the output would be $\mathbf{D} = \{\mathcal{D}_1\} = \{[1]\}$ containing just the first found and thus most probable minimal diagnosis w.r.t. the input DPI. Note that $\mathcal{D}_1 = [1]$ and $\mathcal{D}_2 = [2]$ (which is not computed) have equal probability and whether the one or the other is computed first depends only on the ordering of equally probable (in this case: equal cardinality) nodes in $\mathbf{Q}$. As already mentioned in Section 4.6.2, in this example the most probable diagnosis is equivalent to a minimum cardinality diagnosis since all formula probabilities are equal.

Please notice that the internal "flat" representation used by Algorithm 2 which does not store a tree but only the set of open and closed nodes differs from the standard tree representation [Kal06, FS05, SQJH08, Rei87] we use to depict the hitting set tree *graphically* in Figure 4.2. Whereas within Algorithm 2 a node node stores the set of all the edge labels on the path leading from the root node to node, in the figure we label each node in the tree by the respective label that is computed for this node by the LABEL function, i.e. either by a minimal conflict set, by ✓ or by ×. □

Example 4.9 Recall Example 4.7 which demonstrated how formula fault probabilities are constructed from fault probabilities of syntactical elements for the example DPI depicted by Table 15.3. Now we want to show how the non-interactive KB debugging algorithm given by Algorithm 3 works when these formula probabilities are incorporated.

Suppose the inputs to the algorithm are the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, the function $p(ax)$ for $ax \in \mathcal{K}$ displayed by the rightmost column of Table 4.4 and $auto = false$. Further on, let the user of the debugging algorithm be willing to wait a maximum of one second for an output and let them postulate a minimum of two most probable minimal diagnoses to be returned, e.g. to have at least a second choice if the employed formula probabilities are not perfectly suitable and the most probable diagnosis is not the desired solution. These postulations are expressed by specifying the parameters $n_{\min} = 2$ and $t = 1$ (second). Additionally, assume the user expects the provided probabilities to be sufficiently reasonable such that the
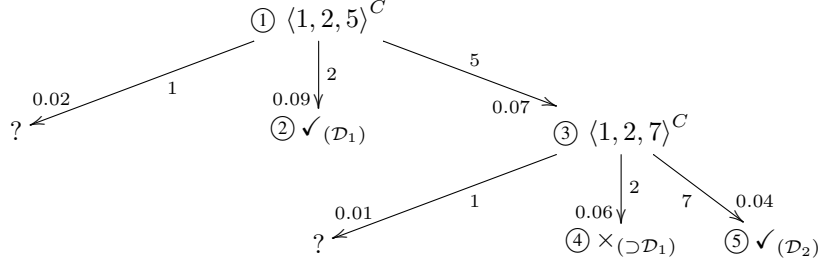
Figure 4.3: Non-interactive KB debugging process with given fault information applied to the DPI given by Table 15.3 with settings $auto = false$, $n_{\min} = 2$, $n_{\max} = 4$ and $t = 1$.

desired diagnosis will be among the best four diagnoses wherefore $n_{\max} = 4$ is chosen. Moreover, let us imagine that the time for each fresh computation of a minimal conflict plus generation of the (unlabeled) successor nodes of this node is 0.4 seconds and the cost of computing any other label of a node is 0.1 seconds.

Then the partial wpHS-tree produced by Algorithm 3 initialized in this way is illustrated by Figure 4.3. The used notation is as described in Example 4.8 with one additional attribute. Namely, each edge is not only labeled by one element of the conflict set from which it goes out, but also by a label $p \in (0, 1)$ that is placed near the arrow head of the arrow that expresses the edge. This label p gives the probability as per $p_{nodes}()$ (cf. Definition 4.9) of the (partial) diagnosis that corresponds to the union of the edge labels along the path from the root to and including the edge that is labeled by p. For example, the label 0.06 of the edge directed at the node number ④ means that the probability of $\{2, 5\}$ is 0.06. Further on, open, i.e. generated, but not yet labeled nodes, are designated by a question mark.

As outlined by the circled numbers ⓘ, as a first action the root node is labeled by the newly computed minimal conflict set $\langle 1, 2, 5 \rangle$, the computation time of which amounts to 0.4. Then, the tree construction proceeds according to the (partial) diagnosis probabilities according to $p_{nodes}()$ computed from the formula probabilities $p(ax)$, $ax \in \mathcal{K}$ provided by the last column of Table 4.4. Therefore, the most probable edge leading away from the root node is labeled next. This already leads to the finding of the first minimal diagnosis $\mathcal{D}_1 = [2]$ after overall computation time of 0.5 seconds. Since $n_{\min} = 2$ diagnoses have not yet been computed and there are still unlabeled open nodes, namely those corresponding to paths $\{1\}$ and $\{5\}$, the algorithm continues the execution by labeling the next best node $\{5\}$ with a probability of 0.07 – as opposed to 0.02 for the other open node $\{1\}$. Since $\{5\}$ is neither a superset of an already computed minimal diagnosis nor a duplicate of another open node nor a diagnosis itself, it must be labeled by some minimal conflict set. Because the already established minimal conflict set $\langle 1, 2, 5 \rangle$ is not disjoint with $\{5\}$, no reuse is possible and QX is called to determine a new minimal conflict set $\langle 1, 2, 7 \rangle$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. All successor nodes of the newly labeled node ③, i.e. the nodes corresponding to the paths $\{1, 5\}$, $\{2, 5\}$ and $\{5, 7\}$, are added to the list $\mathbf{Q}$ of open nodes such that descending order of probabilities is maintained. The resulting queue is then $\mathbf{Q} = [\{2, 5\}, \{5, 7\}, \{1\}, \{1, 5\}]$. As a next step, again the first and thus best open node $\{2, 5\}$ is chosen from $\mathbf{Q}$ and labeled by $\times_{(\supset \mathcal{D}_1)}$ which means that the corresponding path is closed since it is a superset of an already found minimal diagnosis, namely $\mathcal{D}_1 = [2]$. At this point, the overall computation time amounts to 1 second which corresponds to the time limit t. For that reason, the algorithm will go ahead searching for minimal diagnoses only until a minimal number $n_{\min}$ thereof is detected. The node processed next, corresponding to the path $\{5, 7\}$, is then determined to be a minimal diagnosis by the LABEL procedure.

Thus, the output of the algorithm after 1.1 seconds execution time is the set of minimal diagnoses $\mathbf{D} = \{[2], [5, 7]\}$ which is a proper subset of all minimal diagnoses $\mathbf{D}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} = \{[1], [2], [5, 7]\}$.

However, if we assume that the user's intended KB should entail $E \rightarrow G$, for instance, then none of the returned diagnoses can be used to compute a solution KB featuring this entailment when integrated with the background knowledge $\mathcal{B}$. Hence, the true diagnosis $\mathcal{D}_t$ would be missed in this case.

Also, when computing all minimal diagnoses w.r.t. a DPI – if this is even possible in a concrete case due to the computational complexity – and showing them to the user, a user might review just the most probable ones and make a decision on which one to choose only based on these. For instance, [SF10] reported on one DPI where computation of all minimal diagnoses, 1782 in number, is feasible. In such a case it is hard to expect that a user will be willing or will have the time to inspect more than a small fraction of these 1782 diagnoses. The consequence will be a wrong choice of diagnosis in many cases, also because a simple view on a diagnosis will often not lead to the certainty of a user that this one is or is not the desired one. The reason for this is that usually it is too complex for a human brain to perform the necessary mental reasoning to make oneself a picture of the implications of choosing one diagnosis as opposed to another one.

For our example DPI, a user getting the output $\mathbf{D} = \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} = \{[1], [2], [5, 7]\}$ with the computed probabilities $p([1]) = 12\%$, $p([2]) = 60\%$ and $p([5, 7]) = 28\%$ might decide to just inspect the diagnoses that make the most probable 80% fraction of diagnoses. In this case, either $[2]$ or $[5, 7]$ would be selected, which corresponds to a wrong choice in case $E \rightarrow G$ should be entailed be the resulting solution KB after integration with the background KB $\mathcal{B}$. □

Chapter 5

Summary

In this part, we profoundly introduced the topic of knowledge base debugging. We stated necessary properties of knowledge representation languages to be compatible with our approaches, namely that the entailment relation must be monotonic, idempotent and extensive. We gave precise definitions of the problems of KB debugging and parsimonious KB debugging. Both problems assume a given instance of a diagnosis problem (DPI). The former seeks any solution in line with the given requirements whereas the latter seeks a solution that preserves as much formulas as possible of the given faulty KB, i.e. aims at minimal changes. With the *validity of a KB*, a *solution KB*, a *diagnosis* and a *conflict set*, we have characterized central notions that will be extensively used throughout this work. We have studied the relationship between all these notions and proved that solving the problem of parsimonious KB debugging is equivalent to finding a minimal diagnosis w.r.t. a given DPI.

We established the relationship between conflict sets and justifications, a similar notion that is used concurrently to conflict sets in (prevalently DL, OWL or Semantic Web) literature, and provided evidence that conflict sets are the better choice for the debugging problems addressed here. In particular, conflict sets serve the purpose of reducing the search space for minimal diagnoses – minimal hitting sets of all minimal conflict sets – and help a debugging software to focus on the relevant and problematic parts of the faulty KB. A method for the efficient, polynomial time computation of a conflict set was detailed and its correctness was formally proven. Based on this method, we were able to depict a way of computing minimal diagnoses which is based on using a hitting set tree. Such a tree constitutes a systematic way of generating all minimal conflict sets and, in the course of this, also all minimal diagnoses. Depending on the particular situation, the presented algorithm can be configured to compute diagnoses in a predefined order, e.g. most probable diagnoses first or those diagnoses first that are minimally invasive in terms of the changes made to the faulty KB.

Different ways of obtaining and incorporating meta (fault) information into the debugging process were elucidated. Such information, if reasonable, can facilitate and accelerate the debugging process significantly. However, even in the case of the availability of high-quality fault information, we discovered substantial drawbacks of the debugging system presented so far. That is, such a system either chooses automatically a solution (diagnosis) based on the given fault information in a solution space of (generally) exponential size or refers a subset of all solutions, e.g. the most probable solutions, to the user for manual inspection. In the former case, the probability of being presented a solution KB with undesired semantics is very high implying unwanted changes to the faulty KB and unexpected entailments and non-entailments as well as future errors. Such unexpected semantics can be critical or even fatal; one should imagine intelligent medical applications relying on such KBs, for instance. In the latter case, the burden is placed on the user(s) who must mentally anticipate the implications of applying different repairs (using the different submitted diagnoses) to the KB which is practically impossible for human beings both from the

time/effort as well as from the mental perspective. Moreover, it is basically intractable to generate all possible solutions. Hence, it is not even sure that the manually investigated solutions include to correct one (with the postulated semantics).

This leads us to the next part which deals with exactly these issues and proposes a solution.

Part II

Interactive Knowledge Base Debugging

This part is organized as follows:

In Chapter 6, we first discuss how disadvantages of non-interactive KB debugging procedures can be overcome by allowing a user to take part in the debugging process. Next, we define the problem of *interactive static KB debugging* as well as the problem of *interactive dynamic KB debugging* which "naturally" arise from the fact that the DPI in interactive KB debugging is always renewed after a new test case has been specified (a new query has been answered). The former problem searches for a solution KB *w.r.t. the DPI given as input* such that this solution KB satisfies all test cases added during the debugging session and there is no other such solution KB. The latter problem searches for a solution KB *w.r.t. the current DPI* (i.e. the input DPI including all new test cases added throughout the debugging session so far) such that there is no other solution KB w.r.t. the current DPI.

Next, in Chapter 7, the central term of a *query* is specified which constitutes the medium for user interaction. Queries are generated from a set of *leading diagnoses* which is characterized thereafter. The set of leading diagnoses is uniquely partitioned into three subsets by each query. The tuple including these subsets is called *q-partition*. Subsequently, the reader is given some explanations how the q-partition can be interpreted, and how it relates to a query. In fact, we will prove that the notion of a q-partition can serve as a criterion for checking whether a set of logical formulas is a query or not. After that, we will learn that a query exists for any set of (at least two) leading diagnoses which grants that the presented algorithms will definitely be able to come up with a query without the need to impose any restrictions on which (minimal) diagnoses are computed by the diagnosis engine in each iteration.

Chapter 8 shows a method for the generation of (a pool of) set-minimal queries (Algorithm 4) aiming at stressing the interacting user as sparsely as possible, features in-depth discussions of this method's properties, proves its correctness, provides complexity results and gives some illustrating examples. Further on, drawbacks of this method are pointed out and possible solutions are discussed.

Subsequently, Chapter 9 deals with the presentation of the central algorithm of this work which implements an interactive KB debugging system (Algorithm 5). First, an overview of the workflow of interactive KB debugging is given, followed by a more comprehensive detailed specification of the algorithm. Some query selection measures are discussed [RSFF13, SFFR12] and optimization versions of the problems of interactive dynamic and static KB debugging are defined where the goal is to obtain the solution to these problems by asking the user a minimal number of queries. Finally, we prove the correctness of the interactive KB debugging algorithm and provide a discussion of its complexity.

Non-theoretically-oriented readers might well skip Sections 8.2, 8.4, 8.5, 8.7 and 9.4 in this part. Moreover, for the superficially interested reader, it may suffice to concentrate only on Chapter 6 and Sections 7.1, 7.2 and 9.1 in this part.[20]

---

[20]Parts of Part II already appeared in [Rod15].

Chapter 6

Motivation and Problem Definitions

So far, we have learned that the problem of (parsimonious) KB debugging as defined in Problem Definitions 3.1 and 3.2 in Chapter 3 can be solved by investigating minimal diagnoses w.r.t. a given DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. We have seen how minimal diagnoses can be computed, we have introduced a probability space over diagnoses and we have discussed how a-priori probability estimates for diagnoses can be established. Now, assume the situation where a DPI with say 100 minimal diagnoses is given, among which there is one diagnosis $\mathcal{D}$ with highest estimated probability $p(\mathcal{D}) = 10\%$. By the definitions of a diagnosis and a solution KB (Definitions 3.2 and 3.5), each of the 100 diagnoses can be used to formulate a solution KB w.r.t. the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. So, should the system output the solution KB $(\mathcal{K} \setminus \mathcal{D}) \cup U_P$ obtained from $\mathcal{D}$ as the optimal solution? Will a user be satisfied with a likeliness of 90% of being offered a suboptimal solution? What if the diagnoses probabilities are bad estimates and another diagnosis $\mathcal{D}'$ should actually have a probability of 20%?

Why not simply apply Algorithm 3 to show all 100 minimal diagnoses to the user and let them select the preferred one by hand? First, due to the complexity of diagnosis calculation algorithms (cf. Chapter 1), pre-computation of 100 (or, generally, all) minimal diagnoses is usually not tractable within reasonable time. This makes such an approach quite unattractive in an interactive setting. Second, going through large sets of diagnoses can be time-consuming, tedious and error-prone. Third, human beings are normally not capable of (fully) realizing the semantic consequences of deleting a diagnosis from a KB, especially if the KB is large, complex and/or has been created by multiple engineers or automatic systems. Thus, applying a suboptimal diagnosis can result in unexpected entailments or unwanted changes, and thus an incorrect solution KB (incorrect in the sense of the semantics, *not* in the sense of violating given requirements or test cases), which might cause unexpected new faults and contradictions when augmented by new formulas. Consequently, a solution diagnosis is only acceptable if the user has sufficiently scrutinized and approved its semantic effect to the KB.

This leads to the definition of two types of Interactive KB Debugging problems. First, there is the problem of *Interactive Dynamic KB Debugging* which, given an input DPI, aims at the extension of this DPI by new test cases confirmed by a user such that there is only one minimal diagnosis left w.r.t. the extended DPI. Second, we specify the problem of *Interactive Static KB Debugging* which, given an input DPI, aims at the formulation of new test cases confirmed by a user such that these new test cases rule out all but one minimal diagnosis w.r.t. the input DPI.

Problem Definition 6.1 (Interactive Dynamic KB Debugging). *Given a DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, *the task is to find a maximal solution KB* $(\mathcal{K} \setminus \mathcal{D}) \cup U_{P \cup P'}$ *w.r.t. a DPI* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ *such that*

- *$\mathcal{D}$ is the only minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ and*

- *a user has confirmed that each $p' \in P'$ is a positive test case and that each $n' \in N'$ is a negative test case.*

Remark 6.1   The solution of an Interactive Dynamic KB Debugging problem given the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ solves the problem of KB Debugging (Problem Defnition 3.1) as well as the problem of Parsimonious KB Debugging (Problem Defnition 3.2) for the DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$, but in general *not* for the original DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. This is the reason why we term it "dynamic", since a solution is found for a version of the initial DPI that has been extended by test cases.   □

Problem Definition 6.2 (Interactive Static KB Debugging). *Given a DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, *the task is to find a maximal solution KB* $(\mathcal{K} \setminus \mathcal{D}) \cup U_P$ *w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *such that*

- *there are sets of positive test cases P' and negative test cases N' where a user has confirmed that each $p' \in P'$ is a positive test case and that each $n' \in N'$ is a negative test case, and*

- *$\mathcal{D}$ is the only minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that satisfies all positive and negative test cases P' and N', respectively.*

Remark 6.2   The solution of an Interactive Static KB Debugging problem given the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ constitutes a solution to the problem of KB Debugging (Problem Defnition 3.1) as well as to the problem of Parsimonious KB Debugging (Problem Defnition 3.2) *for the original DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, therefore the term "static".   □

Now, we give a more formal definition of a true diagnosis (an informal characterization of which was given in Section 4.6). If sufficiently many new test cases are specified and added to a given DPI such that there is only one remaining minimal diagnosis w.r.t. the input DPI (the input DPI extended by the new test cases) left, then this diagnosis is referred to as the true diagnosis w.r.t. Interactive Static (Dynamic) KB Debugging.

Definition 6.1 (True Diagnosis). *Let $\mathcal{D}_t$ be equal to $\mathcal{D}$ in Problem Definition 9.2 (9.1). Then $\mathcal{D}_t$ is called the* true diagnosis w.r.t. Interactive Static KB Debugging (Interactive Dynamic KB Debugging).

Chapter 7

User Interaction

The idea in interactive KB debugging is to iteratively consult a user asking them to give additional information as regards desired and undesired entailments of the correct KB. Thus, the principle of interactive KB debugging is based on that of *Sequential Diagnosis* which has been suggested by [dKW87] as an iterative way to localize the faulty components (among an initially large set of possibilities) in malfunctioning digital circuits by performing repeated (most informative) measurements. We have shown in our previous works [SF10, SFFR12] how sequential diagnosis can be applied to KBs (ontologies).

In our approach, for the selection of which question (of a pool of possible ones) to ask a user next, an active learning [Set12] approach is applied.[21] *Active Learning* is an iterative supervised machine learning technique in which a learning algorithm is able to interactively query the user to obtain a label for a desired unlabeled instance. In the case of a KB debugging system, an unlabeled instance is a set of logical formulas and the label is whether the conjunction of these formulas should or should not be entailed by the correct KB. Since the learner can choose the instances to be labeled, the number of consultations of an interacting user required to learn a concept (in this case the one solution KB with the desired semantics w.r.t. a given DPI) can often be much lower than the number required in a standard supervised learning setting since the risk that the algorithm must deal with lots of uninformative examples is reduced.

We suppose the user of an interactive KB debugger to be a single person or multiple persons, usually experts of the particular domain the faulty KB is dealing with or authors of the faulty KB. Moreover, we assume the interacting user to be able to answer concrete queries about the intended domain that should be modeled. Otherwise put, we suppose that a user can classify a given logical formula (or a conjunction of logical formulas) as a wanted or unwanted proposition in the intended domain, i.e. as an entailment or non-entailment of the correct domain model. We have already argued in Chapter 1 why this assumption is plausible.

7.1 Queries

In interactive KB debugging, a set of logical formulas Q is presented to the user who should decide whether to assign Q to the set of positive (P) or negative (N) test cases w.r.t. a given DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. In other words, the system asks the user "should the KB you intend to model entail all formulas in Q?". In that, Q is generated by the debugging algorithm in a way that *any* decision of the user

1. invalidates at least one minimal diagnosis (*search space restriction*) and

---

[21] Note that the minimal a-posteriori expected entropy of solution candidate probabilities as a means to select the best next measurement as used in [dKW87] is only one of many possible active learning strategies [Set12].

2. preserves validity of at least one minimal diagnosis (*solution preservation*).

We call a set of logical formulas Q with these properties a *query*. Successive classification of queries as entailments (all formulas in Q must be entailed) or non-entailments (at least one formula in Q must not be entailed) of the correct KB enables gradual restriction of the search space for (minimal) diagnoses. Further on, classification of sufficiently many queries guarantees the detection of a single correct solution diagnosis which can be used to determine a solution KB with the correct semantics w.r.t. a given DPI.[22]

Definition 7.1 (Query). *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ over $\mathcal{L}$ and $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. Then a set of logical formulas $Q \neq \emptyset$ over $\mathcal{L}$ is called a* query *w.r.t.* $\mathbf{D}$ *iff there are diagnoses* $\mathcal{D}, \mathcal{D}' \in \mathbf{D}$ *such that* $\mathcal{D} \notin \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup \{Q\}, N \rangle_R}$ *and* $\mathcal{D}' \notin \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q\} \rangle_R}$. *The set of all queries w.r.t.* $\mathbf{D}$ *and* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *is denoted by* $\mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$.

Remark 7.1   Although Definition 7.1 only postulates that at least one diagnosis in $\mathbf{D}$ is invalidated for whatever answer is given to the query, this implies that, for each answer to the query, there is also a diagnosis that remains valid after adding the corresponding test case to the DPI, as will be shown by Proposition 7.4.   □

So, w.r.t. a set of minimal diagnoses $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, a query Q is a set of logical formulas that rules out at least one diagnosis in $\mathbf{D}$ (and therefore in $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$) as a candidate to formulate a solution KB, regardless of whether Q is classified as a positive or negative test case.

7.2   Leading Diagnoses

Query generation requires a precalculated set of minimal diagnoses $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ that serves as a representative for all minimal diagnoses $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. As already mentioned, computation of the entire set $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ is generally not tractable within reasonable time. Usually, $\mathbf{D}$ is defined as a set of most probable or minimum cardinality diagnoses (cf. Chapter 4). Therefore, $\mathbf{D}$ is called the set of *leading diagnoses w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ [SFFR12].

The leading diagnoses $\mathbf{D}$ are then exploited to determine a query Q the answering of which enables a discrimination between the diagnoses in $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. That is, a subset of $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ which is not "compatible" with the new information obtained by adding the test case Q to P or N is ruled out (see Proposition 7.3 below). For the computation of the subsequent query only a leading diagnoses set $\mathbf{D}_{new}$ w.r.t. the minimal diagnoses still compliant with the new sets of test cases P' and N' is taken into consideration, i.e. $\mathbf{D}_{new} \subseteq \mathbf{D}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R}$.

The number of precomputed leading diagnoses $\mathbf{D}$ affects the quality of the obtained query. The higher $|\mathbf{D}|$, the more representative is $\mathbf{D}$ w.r.t. $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, the more options there are to specify a query in a way that a user can easily comprehend and answer it, and the higher is the chance that a query that eliminates a high rate of diagnoses w.r.t. $\mathbf{D}$ will also eliminate a high rate of all minimal diagnoses $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. The selection of a lower $|\mathbf{D}|$ on the other hand means better timeliness regarding the interaction with a user, first because fewer leading diagnoses might be computed much faster and second because the search space for an "optimal" query is smaller.[23] So, the optimal number of leading diagnoses

---

[22]Correctness of the diagnosis must not be understood as a guarantee that all formulas in the KB which are not in the diagnosis are definitely correct. Instead, correctness must be seen with regard to other diagnoses and with the "Principle of Parsimony" in mind (cf. Section 3.1). That is, all other possible diagnoses are ruled out by a present set of test cases wherefore the single remaining diagnosis is the one that is correct (in comparison with all other incorrect ones). And, there is no evidence (at the time the correct diagnosis is found) that any other formulas in the KB might be faulty. This might change however after new formulas are added to the KB.

[23]Roughly, a query Q is "optimal" if the number of queries that still need to be answered to identify the desired solution KB after Q is added to the (positive or negative) test cases is minimal. "Optimality" of a query can be captured by quantitative information theoretic measures studied in the field of active learning [Set12] that can be used to estimate the quality of a query beforehand, i.e. before an answer to it is known. See Section 9.3 and [RSFF13, SF10, SFFR12] for details.

depends on the complexity of the particular DPI considered. One way to determine a suitable $|\mathbf{D}|$ can be to first define an interval $[n_{\min}, n_{\max}]$ that must comprise $|\mathbf{D}|$ where the upper bound defines the desired number of leading diagnoses and the lower bound the minimally postulated number. Second, the search for minimal diagnoses is run at least as long as it takes to compute $n_{\min}$ diagnoses and at the longest until $n_{\max}$ diagnoses have been found or a timeout t expires that is specified in a manner it enables frequent user interaction. Note that such parameters have already been taken into account in the non-interactive KB debugging Algorithm 2 (see Section 4.7).

7.3 Q-Partitions

Now we introduce the notion of a *q-partition*, a partition of the leading diagnoses set $\mathbf{D}$ induced by a query w.r.t. $\mathbf{D}$. A q-partition will be a helpful instrument in deciding whether a set of logical formulas is a query or not. It will facilitate an estimation of the impact a query answer has in terms of invalidation of minimal diagnoses. And, given fault probabilities, it will enable us to gauge the probability of getting a positive or negative answer to a query.

From now on, given a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and some minimal diagnosis $\mathcal{D}_i$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, we will use the following abbreviation for the solution KB obtained by deletion of $\mathcal{D}_i$ along with the given background knowledge $\mathcal{B}$:

$$\mathcal{K}_i^* := (\mathcal{K} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup U_P \tag{7.1}$$

Definition 7.2 (q-Partition[24]). *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI over $\mathcal{L}$, $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. Further, let Q be a set of logical formulas over $\mathcal{L}$ and*

- $\mathbf{D}^+(Q) := \{ \mathcal{D}_i \in \mathbf{D} \mid \mathcal{K}_i^* \models Q \}$,

- $\mathbf{D}^-(Q) := \{ \mathcal{D}_i \in \mathbf{D} \mid \exists x \in R \cup N : \mathcal{K}_i^* \cup Q \text{ violates } x \}$,

- $\mathbf{D}^0(Q) := \mathbf{D} \setminus (\mathbf{D}_j^+ \cup \mathbf{D}_j^-)$.

Then $\langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q) \rangle$ is called a q-partition *iff Q is a query w.r.t. $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Remark 7.2 The set $\mathbf{D}^-(Q)$ contains exactly those diagnoses $\mathcal{D}_i \in \mathbf{D}$ where $\mathcal{K} \setminus \mathcal{D}_i$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P \cup \{Q\}, N \rangle$ (cf. Definition 3.3). $\square$

Proposition 7.1. *For each query Q w.r.t. some $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ it holds that $\langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q) \rangle$ is a partition of $\mathbf{D}$.*

Proof. First, by definition of $\mathbf{D}^0(Q)$, we have that $\mathbf{D}^+(Q) \cup \mathbf{D}^-(Q) \cup \mathbf{D}^0(Q) = \mathbf{D}$, $\mathbf{D}^+(Q) \cap \mathbf{D}^0(Q) = \emptyset$ and $\mathbf{D}^-(Q) \cap \mathbf{D}^0(Q) = \emptyset$. Second, $\mathbf{D}^+(Q) \cap \mathbf{D}^-(Q) = \emptyset$ since $\mathcal{K}_i^* \models Q_j$ and $\exists x \in R \cup N : (\mathcal{K}_i^* \cup Q_j \text{ violates } x)$ imply by idempotency of $\mathcal{L}$ that $\mathcal{K}_i^*$ violates some $x \in R \cup N$ which is a contradiction to $\mathcal{D}_i$ being a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Thus, each diagnosis in $\mathbf{D}$ is an element of exactly one set of $\mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q)$ which is equivalent to the statement of the proposition. $\square$

Remark 7.3 In fact, Proposition 7.1 holds for any set $\mathbf{D} \subseteq \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, i.e. for any subset of *all* diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. This can be easily seen from the proof of Proposition 7.1 which does not require minimality of diagnoses. That is, any set of diagnoses w.r.t. a DPI is partitioned into the three sets $\mathbf{D}^+(Q), \mathbf{D}^-(Q)$ and $\mathbf{D}^0(Q)$ as per Definition 7.2 by a query Q w.r.t. this DPI. $\square$

---

[24]In existing literature, e.g. [SFFR12, RSFF13, SF10], a q-partition is often simply referred to as partition. We call it q-partition to emphasize that not each partition of $\mathbf{D}$ into three sets is necessarily a q-partition.

Proposition 7.2. *For each query Q w.r.t. some $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ there is one and only one partition* $\langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q) \rangle$.

Proof. The existence of a partition $\mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q)$ follows directly from Proposition 7.1. Assume there are two different partitions $\langle \mathbf{D}_1^+(Q), \mathbf{D}_1^-(Q), \mathbf{D}_1^0(Q) \rangle$ and $\langle \mathbf{D}_2^+(Q), \mathbf{D}_2^-(Q), \mathbf{D}_2^0(Q) \rangle$. Then, (a) $\mathbf{D}_1^+(Q) \neq \mathbf{D}_2^+(Q)$ or (b) $\mathbf{D}_1^-(Q) \neq \mathbf{D}_2^-(Q)$ or (c) $\mathbf{D}_1^0(Q) \neq \mathbf{D}_2^0(Q)$ must hold. If (a) is true, then there is one diagnosis $\mathcal{D}_i \in \mathbf{D}$ such that $\mathcal{K}_i^* \models Q$ and $\mathcal{K}_i^* \not\models Q$ – a contradiction. If (b) is true, then there is one diagnosis $\mathcal{D}_i \in \mathbf{D}$ such that $\mathcal{K}_i^* \cup Q$ violates some $x \in R \cup N$ and $\mathcal{K}_i^* \cup Q$ does not violate any $y \in R \cup N$ – a contradiction. If (c) is true, then $(\mathbf{D}_1^+(Q) \cup \mathbf{D}_1^-(Q)) \neq (\mathbf{D}_2^+(Q) \cup \mathbf{D}_2^-(Q))$ which implies that either (a) or (b) must be true. $\square$

Due to the uniqueness of a q-partition $\langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q) \rangle$ for a query Q, we denote this q-partition by $\mathfrak{P}(Q)$. As a consequence of Definition 7.2 and Proposition 7.2, a query Q is a set of common entailments of KBs $\mathcal{K}_i^*$, each resulting from the deletion of a single minimal diagnosis $\mathcal{D}_i \in \mathbf{D}^+(Q)$ from $\mathcal{K}$.

Corollary 7.1. *For each query $Q \in \mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ there is a set of minimal diagnoses $\mathbf{D}^+(Q) \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ as defined by Definition 7.2 such that $Q \subseteq \{e \mid \forall \mathcal{D}_i \in \mathbf{D}^+(Q) : \mathcal{K}_i^* \models e\}$.*

7.4   Interpretation of Q-Partitions

Since $\mathcal{K}_i^*$ corresponds to the solution KB (along with $\mathcal{B}$) obtained under the assumption that $\mathcal{D}_t = \mathcal{D}_i$, i.e. the true diagnosis (cf. Definition 6.1) corresponds to $\mathcal{D}_i$, the sets $\mathbf{D}^+(Q)$ and $\mathbf{D}^-(Q)$ can be interpreted as those leading diagnoses that predict the classification of Q as a positive and negative test case, respectively. In other words, if the true diagnosis $\mathcal{D}_t$ is in $\mathbf{D}^+(Q)$, then the true solution KB $\mathcal{K}_t^*$ entails Q by Definition 7.2. Therefore the user will answer Q positively (cf. Definition 6.1). If, conversely, $\mathcal{D}_t$ is in $\mathbf{D}^-(Q)$, then the true solution KB $\mathcal{K}_t^*$ would be invalidated if Q was answered positively, since $\mathcal{K}_t^* \cup Q = (\mathcal{K} \setminus \mathcal{D}_t) \cup \mathcal{B} \cup U_{P \cup \{Q\}}$ violates some $x \in R \cup N$ and thus $\mathcal{K} \setminus \mathcal{D}_t$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P \cup \{Q\}, N \rangle_R$, which implies that $\mathcal{D}_t$ is not a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q\}, N \rangle_R$ according to Proposition 3.2. Hence, the user will answer Q negatively (cf. Definition 6.1). Diagnoses in $\mathbf{D}^0(Q)$ on the other hand neither predict $Q \in P$ nor $Q \in N$. This means that we do not know how the user will answer a query Q for which the true diagnosis $\mathcal{D}_t$ is in $\mathbf{D}^0(Q)$. In this case, for any answer to Q, the true diagnosis $\mathcal{D}_t$ is in the set of minimal diagnoses w.r.t. the new DPI including Q as a test case. To summarize: If the true diagnosis $\mathcal{D}_t$ is an element of $\mathbf{D}^+(Q)$ ($\mathbf{D}^-(Q)$), then Q will be answered positively (negatively).

Conversely, this means that a q-partition $\mathfrak{P}(Q)$ gives a prior indication which leading diagnoses would be invalidated by a user's answer. Diagnoses in $\mathbf{D}^+(Q)$ are invalidated by the classification $Q \in N$, and diagnoses in $\mathbf{D}^-(Q)$ in case of $Q \in P$. Diagnoses in $\mathbf{D}^0(Q)$ can never be invalidated by an answer to Q. Thus, intuitively, queries with $\mathbf{D}^0(Q) = \emptyset$ are preferable over other queries (as per the information provided by the set of leading diagnoses $\mathbf{D}$) as the number of (definitely) eliminated diagnoses in $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ should be maximized.

The following proposition is a direct consequence of Corollary 3.3 and explicates the impact of the addition of a test case to a DPI regarding the set of minimal diagnoses for this DPI.

Proposition 7.3. *Let Q be a query w.r.t. $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ and let the answer of a user to Q be $u(Q) \in \{true, false\}$.*

If $u(Q) = true$, then $\mathcal{D}_i \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q\}, N \rangle_R$ iff $\mathcal{K} \setminus \mathcal{D}_i$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P \cup \{Q\}, N \rangle_R$.

In other words, both of the following conditions must hold:

$$\forall r \in R \;:\; \mathcal{K}_i^* \cup Q \text{ does not violate } r$$
$$\forall n \in N \;:\; \mathcal{K}_i^* \cup Q \not\models n$$

If $u(Q) = false$, then $\mathcal{D}_i \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q\} \rangle_R$ iff $\mathcal{K} \setminus \mathcal{D}_i$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \cup \{Q\} \rangle_R$.

In other words, both of the following conditions must hold:

$$\forall r \in R \;:\; \mathcal{K}_i^* \text{ does not violate } r$$
$$\forall n \in (N \cup \{Q\}) \;:\; \mathcal{K}_i^* \not\models n$$

Remark 7.4 From Proposition 7.3 and Definition 7.2 it is easy to see that at least $\mathcal{D}_i \in \mathbf{D}^-(Q) \subset \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ are eliminated by a positive answer to Q. Namely, $\mathbf{D}^-(Q)$ comprises exactly those diagnoses $\mathcal{D}_i$ that imply the violation of some $r \in R$ or the entailment of some $n \in N$ if Q is added to $\mathcal{K}_i^*$. On the other hand, at least $\mathcal{D}_i \in \mathbf{D}^+(Q) \subset \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ are discarded if $u(Q) = false$ as all diagnoses in $\mathbf{D}^+(Q)$ entail Q which must not be entailed.

Note that, in general, the addition of a query to the test cases of a DPI causes not only an invalidation of some leading minimal diagnoses in $\mathbf{D}$, but also the elimination of minimal diagnoses that have not even been computed yet. On the other hand, an added test case might also introduce new *minimal* diagnoses, i.e. ones that were no minimal diagnoses before this test case was added. However, the newly obtained DPI after the addition of any new test case can only exhibit a reduced set of *all* (i.e. minimal and non-minimal) diagnoses compared with the DPI before the test case was added (we will prove this result by Proposition 12.3). □

7.5 The Relation between a Query and Its Q-Partition

The following proposition shows the relationship between a query and its q-partition and provides a criterion that enables to check whether a set of logical formulas is a query w.r.t. some set of leading diagnoses or not.

Proposition 7.4. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI over $\mathcal{L}$ and $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. Then a set of logical formulas $Q \neq \emptyset$ over $\mathcal{L}$ is a query w.r.t. $\mathbf{D}$ iff $\mathbf{D}^+(Q) \neq \emptyset$ and $\mathbf{D}^-(Q) \neq \emptyset$.*

Proof. "$\Leftarrow$": If $\mathbf{D}^+(Q) \neq \emptyset$ and $\mathbf{D}^-(Q) \neq \emptyset$ holds, then a non-empty set of diagnoses $\mathbf{D}^-(Q)$ $(\mathbf{D}^+(Q))$ becomes invalid for positive (negative) answer to Q. So, Q is a query.

"$\Rightarrow$": If Q is a query, then there are diagnoses $\mathcal{D}, \mathcal{D}' \in \mathbf{D}$ such that $\mathcal{D} \notin \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup \{Q\}, N \rangle_R}$ and $\mathcal{D}' \notin \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q\} \rangle_R}$. Consequently, $\mathcal{D} \in \mathbf{D} \setminus \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup \{Q\}, N \rangle_R}$ and $\mathcal{D}' \in \mathbf{D} \setminus \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q\} \rangle_R}$ holds. But, as the diagnoses in $\mathbf{D} \setminus \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup \{Q\}, N \rangle_R}$ are exactly the diagnoses in $\mathbf{D}$ that become invalid by the positive answer to Q, we obtain $\mathcal{D} \in \mathbf{D}^-(Q)$. The argumentation for $\mathcal{D}' \in \mathbf{D}^+(Q)$ is analogous. Hence, $\mathbf{D}^+(Q) \neq \emptyset$ and $\mathbf{D}^-(Q) \neq \emptyset$. □

Corollary 7.2. *Let $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. Then, for each q-partition $\mathfrak{P}(Q) = \langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q) \rangle$ w.r.t. $\mathbf{D}$ it holds that $\mathbf{D}^+(Q) \neq \emptyset$ and $\mathbf{D}^-(Q) \neq \emptyset$.*

Proof. Follows from Definition 7.2 which grants the existence of a query for any q-partition and Proposition 7.4 which states that neither $\mathbf{D}^+(Q)$ nor $\mathbf{D}^-(Q)$ must be empty sets for any query. □

So, by Proposition 7.4, a query not only eliminates at least one leading diagnosis, but also leaves at least one leading diagnosis valid. Therefore, an admissible DPI can never get non-admissible by adding a query to the positive or negative test cases.

Corollary 7.3. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be an admissible DPI, $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ and $Q \in \mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$.
Then $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q\}, N \rangle_R$ as well as $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q\} \rangle_R$ are admissible DPIs.*

Proof. Assume that $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q\}, N \rangle_R$ is non-admissible. Then there is no valid diagnosis for this
DPI. Since $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is an admissible DPI, this means that Q invalidates each diagnosis $\mathcal{D} \in$
$\mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \supseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \supset \mathbf{D}$. By Proposition 7.4, this is a contradiction to the fact that Q
is a query. The argumentation for $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q\} \rangle_R$ is analogue. $\qquad \square$

This means in particular that a query can never contain a conflict set or result in a violation of some
requirement $r \in R$ when added to $\mathcal{B} \cup U_P$ (cf. Proposition 3.4).

7.6   Existence of Queries

For any set of at least two leading minimal diagnoses the existence of a query is guaranteed, as the next
proposition and corollary show. In particular, this implies that for arbitrary two minimal diagnoses $\mathcal{D}, \mathcal{D}'$
w.r.t. a DPI there is a query Q that enables to differentiate between $\mathcal{D}$ and $\mathcal{D}'$, i.e. exactly one of these
diagnoses is invalidated by each answer to Q.

Proposition 7.5. *Let $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ with $|\mathbf{D}| \geq 2$ and $U_{\mathbf{D}}$ be the union of all diagnoses in $\mathbf{D}$.
Then*

   (I) $Q := (U_{\mathbf{D}} \setminus \mathcal{D}_i)$ is a query w.r.t. $\mathbf{D}$ for arbitrary $\mathcal{D}_i \in \mathbf{D}$ and

   (II) $\mathfrak{P}(Q) = \langle \{\mathcal{D}_i\}, \mathbf{D} \setminus \{\mathcal{D}_i\}, \emptyset \rangle$.

Proof. **Ad (I):** Assume that Q is not a query. Then either (1) $Q = \emptyset$ or (2) $\mathbf{D}^+(Q) = \emptyset$ or (3) $\mathbf{D}^-(Q) = \emptyset$.
In the following we prove that neither (1) nor (2) nor (3) can hold.
   (1): $Q = \emptyset$ means that $\mathcal{D}_i \supseteq U_{\mathbf{D}}$. Since any diagnosis $\mathcal{D}$ in $\mathbf{D}$ is a subset of $U_{\mathbf{D}}$, this implies that for
each $\mathcal{D} \in \mathbf{D}$, $\mathcal{D} \subseteq \mathcal{D}_i$ holds. As $|\mathbf{D}| \geq 2$ is assumed, there is a $\mathcal{D}_k \neq \mathcal{D}_i \in \mathbf{D}$ for which this property
holds. This, however, is a contradiction to the minimality of diagnosis $\mathcal{D}_i$.
   (2): $\mathbf{D}^+(Q) = \emptyset$ cannot hold, since $(\mathcal{K} \setminus \mathcal{D}_i) \supseteq (U_{\mathbf{D}} \setminus \mathcal{D}_i)$ and $U_{\mathbf{D}} \setminus \mathcal{D}_i \models Q$ by monotonicity of
description logics imply that $\mathcal{K}_i^* = (\mathcal{K} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup U_P \models Q$. Hence, there is at least one diagnosis,
namely $\mathcal{D}_i$, in $\mathbf{D}^+(Q)$.
   (3): To prove that $\mathbf{D}^-(Q) \neq \emptyset$, we must show that there is a diagnosis $\mathcal{D} \in \mathbf{D}$ such that $Y := (\mathcal{K} \setminus$
$\mathcal{D}) \cup \mathcal{B} \cup U_P \cup Q = (\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup U_P \cup (U_{\mathbf{D}} \setminus \mathcal{D}_i)$ is incoherent. However, $(\mathcal{K} \setminus \mathcal{D}) \cup (U_{\mathbf{D}} \setminus \mathcal{D}_i) = \mathcal{K} \setminus (\mathcal{D} \cap \mathcal{D}_i)$
by distributive and De Morgan laws which yields $Y = \mathcal{K} \setminus (\mathcal{D} \cap \mathcal{D}_i) \cup \mathcal{B} \cup U_P$. But, $\mathcal{D} \cap \mathcal{D}_i \subset \mathcal{D}$ must
hold as $\mathcal{D} \not\subseteq \mathcal{D}_i$ by the subset-minimality of $\mathcal{D}_i$ whereby $\mathcal{D}$ must comprise a formula $ax \notin \mathcal{D}_i$. Hence,
$Y \supset (\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup U_P$ is incoherent by subset-minimality of $\mathcal{D}$.
   Ad (II): We already know that $\mathcal{D}_i \in \mathbf{D}^+(Q)$ by (2). Since $\mathcal{D} \in \mathbf{D}$ in (3) can be chosen arbitrarily,
we obtain that $\mathcal{D} \in \mathbf{D}^-(Q)$ for all diagnoses $\mathcal{D} \in \mathbf{D} \setminus \{\mathcal{D}_i\}$. $\qquad \square$

We immediately obtain a lower bound for the number of queries by Proposition 7.5:

Corollary 7.4. *Let $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ with $|\mathbf{D}| > 1$. Then a lower bound for the number of queries
w.r.t. $\mathbf{D}$ is $|\mathbf{D}|$.*

Remark 7.5   Notice that the preceding proposition and corollary require a set of *minimal* diagnoses.
This means that subset-minimality of diagnoses is a necessary prerequisite for guaranteeing the possibility
of discrimination between diagnoses. In other words, interactive debugging by means of (some or only)
non-minimal diagnoses cannot be proven to work correctly (without making any further assumptions).
$\qquad \square$

Chapter 8

Query Generation

In this chapter we want to describe, discuss and prove the correctness of methods for the generation of queries which takes place in each iteration of an interactive KB debugging algorithm after a set of leading diagnoses has been determined. With Algorithm 4, similar versions of which can be found in [SFFR12, RSFF13], we present a way to compute a pool $\mathbf{QP}$ of queries and associated q-partitions w.r.t. a set of leading diagnoses $\mathbf{D}$ and a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. The generation of this pool $\mathbf{QP}$ is the first stage of the query computation function used in the interactive debugging algorithm (Algorithm 5) presented below. In a second stage, one particular query that meets certain criteria such as maximum expected information gain is selected from $\mathbf{QP}$ (see Section 9.3).

Before we give a description of Algorithm 4, let us have a look at some example by which we want to demonstrate the principle how a query w.r.t. some set of leading diagnoses for a DPI can be constructed. This should give the reader a first idea and an intuition of how the presented algorithm works.

Example 8.1 Consider the example FOL DPI given by Table 15.2. The set of minimal conflict sets $\mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} = \{\mathcal{C}_1, \mathcal{C}_2\} = \{\langle 1, 3, 4 \rangle, \langle 1, 2, 3, 5 \rangle\}$ (like in previous examples, formulas ax_i in Table 15.2 are sometimes referred to just by their number i if it is clear from the context what is meant). Let the set of leading diagnoses be the set of all minimal diagnoses, i.e. $\mathbf{D} = \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\} = \{[1], [3], [4, 5], [2, 4]\}$. To enable a better understanding of this example, we first analyze why $\mathcal{C}_1$ and $\mathcal{C}_2$ are minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Why is $\mathcal{C}_1$ a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$? In the following we underline the formulas ax_i and relevant parts of these formulas used in the derivation of the conflict set. First, there is the background KB $\mathcal{B}$ including $\underline{a_1(w)}$ and $\underline{a_1(u)}$. Due to $\underline{ax_1}$, by substitution of X by w (written as X/w), we obtain $a_2(w), m_1(w)$ and $\underline{m_2(w)}$ from $a_1(w)$. Likewise, we can derive $a_2(u), m_1(u)$ and $\underline{m_2(u)}$ from $a_1(u)$ by X/u. Substituting X by w in $\underline{ax_3}$ yields $m_1(w) \to \neg a(w) \wedge b(w)$. Thus, we obtain $\neg a(w)$. A substitution of X by u in $\underline{ax_4}$ results in $m_2(u) \to (\forall Y s(u, Y) \to a(Y)) \wedge d(u)$. By Y/w, we have $\underline{m_2(u) \to (s(u, w) \to a(w)) \wedge d(u)}$. Since $m_2(u)$ has already been deduced from the background formula $a_1(u)$ and $\underline{s(u, w)}$ is a background formula as well, we can conclude $a(w)$ from ax_4. All in all, we have derived $\neg a(w)$ and $a(w)$, i.e. an inconsistency, by means of $\mathcal{B}$ and $\mathcal{C}_1$ (and U_P which is the empty set) wherefore $\mathcal{C}_1$ is a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ by Definition 4.1. The minimality of $\mathcal{C}_1$ can be easily verified by the way we derived that it is a conflict set; namely, leaving out any of the formulas ax_1, ax_3 or ax_4 does not allow to derive an inconsistency or incoherency (note that the set of negative test cases N is empty).

Why is $\mathcal{C}_2$ a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$? We argue as follows to deduce the inconsistency

99

i	ax_i	$\mathcal{K}$	$\mathcal{B}$
1	$\forall X a_1(X) \;\rightarrow\; a_2(X) \wedge m_1(X) \wedge m_2(X)$	•	
2	$\forall X a_2(X) \;\rightarrow\; \neg(\exists Y s(X,Y) \wedge m_3(Y)) \wedge \exists Z s(X,Z) \wedge m_2(Z)$	•	
3	$\forall X m_1(X) \;\rightarrow\; \neg a(X) \wedge b(X)$	•	
4	$\forall X m_2(X) \;\rightarrow\; (\forall Y s(X,Y) \rightarrow a(Y)) \wedge d(X)$	•	
5	$\forall X m_3(X) \;\leftrightarrow\; b(X) \vee c(X)$	•	
6	$a_1(w)$		•
7	$a_1(u)$		•
8	$s(u,w)$		•
i	$p_i \in P$		
×	×		
i	$n_i \in N$		
×	×		
i	$r_i \in R$		
1	consistency		
2	coherency		

Table 8.1: First-Order Logic Example DPI

responsible for $\mathcal{C}_2$ to be a conflict set (the relevant implications and used formulas are again underlined):

$$(1):\; a_1(w) \in \mathcal{B}:\quad \underline{a_1(w)}$$
$$(2):\; X/w \text{ in } \underline{ax_1}:\quad \underline{a_1(w)} \;\rightarrow\; a_2(w) \wedge \underline{m_1(w)} \wedge m_2(w)$$
$$(3):\; X/w \text{ in } \underline{ax_3}:\quad \underline{m_1(w)} \;\rightarrow\; \neg a(w) \wedge \underline{b(w)}$$
$$(4):\; \underline{ax_5} \text{ and } X/w:\quad \underline{b(w)} \;\rightarrow\; \underline{m_3(w)}$$
$$(5):\; (1)-(4):\quad \underline{m_3(w)}$$
$$(6):\; a_1(u) \in \mathcal{B}:\quad \underline{a_1(u)}$$
$$(7):\; X/u \text{ in } \underline{ax_1}:\quad \underline{a_1(u)} \;\rightarrow\; \underline{a_2(u)} \wedge m_1(u) \wedge m_2(u)$$
$$(8):\; X/u \text{ in } \underline{ax_2}:\quad \underline{a_2(u)} \;\rightarrow\; \underline{\neg(\exists Y s(u,Y) \wedge m_3(Y))}$$
$$\wedge\, (\exists Z s(u,Z) \wedge m_2(Z))$$
$$(9):\; (6)-(8):\quad \underline{\neg(\exists Y s(u,Y) \wedge m_3(Y))}$$
$$(10):\; s(u,w) \in \mathcal{B}:\quad \underline{s(u,w)}$$
$$(11):\; (5) \text{ and } (10):\quad \underline{\exists Y s(u,Y) \wedge m_3(Y)}$$
$$(9) \text{ and } (11):\quad \text{\textlightning} \qquad \square$$

Minimality of $\mathcal{C}_2$ can again be verified by observing that, given any formula of $\mathcal{C}_2$ is left out, no inconsistency or incoherency can be derived.

Now we show how to construct a query manually. As suggested by Definition 7.2 and Proposition 7.4 and discussed in Section 7.5, an obvious way of generating a query w.r.t. $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is via the notion of a q-partition. Definition 7.2 states that Q is a set of common entailments of KBs $\mathcal{K}_i^*$ (Formula 7.1) where $\mathcal{D}_i \in \mathbf{D}^+(Q)$, a subset of $\mathbf{D}$. Hence, a first step towards query computation is to choose some non-empty subset $\mathbf{S}$ of the leading diagnoses $\mathbf{D}$ which we will call the *seed* for query generation. For our manual construction, let $\mathbf{S} = \{\mathcal{D}_3, \mathcal{D}_4\} = \{[4,5], [2,4]\}$. For each of the diagnoses

$\mathcal{D}_i$ in $\mathbf{S}$, we assemble the KB $\mathcal{K}_i^*$ and use a reasoning engine to obtain a set of entailments $E_{\mathcal{D}_i}$ of $\mathcal{K}_i^*$. For $\mathcal{D}_3$ we obtain $\mathcal{K}_3^* := \{1,2,3,4,5\} \setminus \{4,5\} \cup \{6,7,8\} \cup \{\} = \{1,2,3,6,7,8\}$. Similarly, we compute $\mathcal{K}_4^* = \{1,3,5,6,7,8\}$.

Suppose that the reasoner invoked by the used GETENTAILMENTS function produces only entailments of the type $\forall X\, p_1(X) \to p_2(X)$ for predicate names p_1, p_2 and of the type $p(a)$ where p is a predicate name and a is a constant (cf. Remark 2.3). For this purpose, DL and OWL reasoners, respectively, such as Pellet [SPG$^+$07], HermiT [SMH08], FaCT++ [TH06] or KAON2[25] could be used with their classification and realization reasoning services. The reason why this is possible can be realized after a short analysis of the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given by Table 15.2. For, this DPI can be translated to DL similarly as demonstrated in Example 2.1. All the mentioned reasoners can deal with the expressivity of the resulting DL language.

Then, we obtain the sets $E_{\mathcal{D}_3}$ and $E_{\mathcal{D}_4}$, i.e. the sets of entailments of $\mathcal{K}_3^*$ and $\mathcal{K}_4^*$, respectively, as depicted by Table 8.2. The set of common entailments Q, i.e. $Q = E_{\mathcal{D}_3} \cap E_{\mathcal{D}_4}$ is then the set containing all elements in the rows of Table 8.2 that are above the dashed line.

Notice at this point that the set $\{a_1(w), a_1(u), s(u,w)\} = \mathcal{B}$ does not need to be computed or, respectively, included in Q since none of these formulas can serve to discriminate between diagnoses (which is the only aim of a query). The simple reason for this is that $\mathcal{K}_i^*$ for *each* $\mathcal{D}_i \in \mathbf{D}$ comprises these formulas and thus each $\mathcal{K}_i^*$ entails these formulas by the extensiveness of FOL (cf. Chapter 2). Since entailed by each potential solution KB $\mathcal{K}_i^*$, these formulas cannot yield a violation of any requirements or test cases since none of the KBs $\mathcal{K}_i^*$ violates any requirements or test cases (follows from Definitions 3.5 and 3.2).

Continuing with our query construction, we know by Proposition 7.4 that Q is a query w.r.t. $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ iff $\mathbf{D}^+(Q) \neq \emptyset$ and $\mathbf{D}^-(Q) \neq \emptyset$. Whereas it is trivial that the former condition is met since $\mathbf{D}^+(Q)$ contains (at least) the two diagnoses $\mathcal{D}_3$ and $\mathcal{D}_4$ that we used to compute Q (cf. Definition 7.2), we still need to verify whether the latter condition is actually satisfied for Q. To this end, as per Definition 7.2, we must simply find some diagnosis $\mathcal{D}_j$ in $\mathbf{D} \setminus \mathbf{S} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\} \setminus \{\mathcal{D}_3, \mathcal{D}_4\} = \{\mathcal{D}_1, \mathcal{D}_2\}$ such that $\mathcal{K}_j^* \cup Q$ violates some $x \in N \cup R$, i.e. whether some negative test case is entailed or whether this KB is incoherent or inconsistent. So, we start with $\mathcal{D}_1$, i.e. we examine $(\mathcal{K} \setminus \mathcal{D}_1) \cup \mathcal{B} \cup P \cup Q = \{1,2,3,4,5\} \setminus \{1\} \cup \{6,7,8\} \cup \{\} \cup Q = \{2,3,4,5,6,7,8\} \cup Q$.

And, indeed, we are able to prove an inconsistency for this KB. To see that, verify that by X/w in $e_2 \in Q$ (see Table 8.2) and $a_1(w) = ax_6 \in \mathcal{K}_1^*$ we can derive $m_1(w)$ which lets us conclude $\neg a(w)$ by the substitution of X by w in $ax_3 \in \mathcal{K}_1^*$. On the other hand, we obtain $a(w)$ by X/u in $e_3 \in Q$, $\{X/u, Y/w\}$ in $ax_4 \in \mathcal{K}_1^*$ and $s(u,w) = ax_8 \in \mathcal{K}_1^*$ as shown in the explanation for conflict set $\mathcal{C}_1$ above. Thus, $\mathcal{D}_1 \in \mathbf{D}^-(Q)$.

That is, we have just proven that Q is de facto a query w.r.t. $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. And this, although we have not yet assigned each leading diagnosis to the respective set of the q-partition of Q. In a situation where just any query shall be asked to the user, this would suffice, and the query could be presented to the interacting user.

However, in case a "best" query according to some criterion shall be determined from a set of different competing queries, usually the computation of the full q-partition of each competing query is required. This is due to the fact that the q-partition provides information about several properties of queries that are considered by common query selection techniques (for details see Section 9.3). So, let us complete the q-partition for our query Q by investigating $\mathcal{K}_2^* \cup Q = \{1,2,4,5,6,7,8\} \cup Q$. Also in this case we can derive an inconsistency which can be easily realized by reconsidering the argumentation why $\mathcal{C}_2$ is a conflict set above and by using $e_4 \in Q$ instead of $ax_3 \notin \mathcal{K}_2^* \cup Q$. That means, the final q-partition $\mathfrak{P}(Q)$ for Q is given by $\langle \{\mathcal{D}_3, \mathcal{D}_4\}, \{\mathcal{D}_1, \mathcal{D}_2\}, \emptyset \rangle$.

The next question that arises directly from the proofs that $\mathcal{D}_3, \mathcal{D}_4 \in \mathbf{D}^-(Q)$ is whether there is a (set-minimal) subset $Q_{\min}$ of Q such that $Q_{\min}$ preserves the discrimination properties of Q, i.e. the

---

[25]http://kaon2.semanticweb.org/

q-partition $\mathfrak{P}(Q_{\min}) = \mathfrak{P}(Q)$. In fact, the answer is yes for the query Q we computed, but also for the majority of other cases. This is a simple consequence of using the reasoning engine as a black-box which suggests a strategy we pursued in our query construction which relies on a precomputation of entailments and a final minimization part. Sticking to this black-box concept however does not allow to use some customized reasoning procedure that pointedly returns a set of common entailments Q for a set of diagnoses $\mathbf{S} \subset \mathbf{D}$ where all formulas in Q are necessary for a requirement or test case violation, respectively, of KBs $\mathcal{K}_j^*$ for diagnoses in $\mathbf{D} \setminus \mathbf{S}$.

What militates for such a black-box approach is the generality and independence of a particular logic (for which an adequate glass-box reasoner exists), the easier implementation of the debugging system and potential performance issues with a glass-box approach [KPSH05]. For a black-box algorithm to work, only a reasoner implementing a sound and complete inference procedure for the used logic $\mathcal{L}$ must be available.

In general, there is more than one minimized version of a query that preserves the q-partition. Theoretically, the number of such minimal queries w.r.t. one q-partition can be exponential in the size of the initially computed query that is provided as an input to the minimization procedure. For our query Q, for instance,

$$
\begin{aligned}
Q_{\min,1} &= \{a_2(u), b(w)\} = \{e_7, e_{12}\}, \\
Q_{\min,2} &= \{\forall X a_1(X) \to a_2(X), b(w)\} = \{e_1, e_{12}\}, \\
Q_{\min,3} &= \{\forall X a_1(X) \to a_2(X), \\
&\qquad \forall X a_1(X) \to m_1(X), \\
&\qquad \forall X m_1(X) \to b(X)\} = \{e_1, e_2, e_4\} \quad \text{and} \\
Q_{\min,4} &= \{\forall X a_1(X) \to m_1(X), \\
&\qquad \forall X a_1(X) \to m_2(X), \\
&\qquad \forall X m_1(X) \to b(X)\} = \{e_2, e_3, e_4\}
\end{aligned}
$$

are set-minimal, q-partition preserving subqueries. Namely, each of the sets $Q_{\min,1}$, $Q_{\min,2}$ and $Q_{\min,3}$ together with $\{2, 5, 6, 7, 8\}$ implies an inconsistency since $m_3(w)$ and $\neg m_3(w)$ can be derived and $\{2, 5, 6, 7, 8\} \subseteq \mathcal{K}_1^*$ and $\{2, 5, 6, 7, 8\} \subseteq \mathcal{K}_2^*$. $\{e_2, e_3\} \subset Q_{\min,4}$ yields an inconsistency when added to $\mathcal{K}_1^*$, i.e. $a(w)$ and $\neg a(w)$ are entailed, and $\{e_4\} \subset Q_{\min,4}$ merged with $\mathcal{K}_2^*$ yields an inconsistency, i.e. the derivation of $m_3(w)$ and $\neg m_3(w)$. In order not to overwhelm the user we would of course ask them such a minimized version of a query rather than the full query that contains plenty of irrelevant formulas.

An example of a seed $\mathbf{S}$ that does not lead to the discovery of a query is $\mathbf{S} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$ since the set of common entailments $E_{\mathcal{D}_1} \cap E_{\mathcal{D}_2} \cap E_{\mathcal{D}_3} = \emptyset$. Note that this holds when all $E_{\mathcal{D}_i}$ contain only entailments of the types we specified above. For other types of entailments, i.e. a different specification of the GETENTAILMENTS function, this might no longer hold. $\qquad\square$

8.1   Generation of a Pool of Queries

The main function GETPOOLOFQUERIES of Algorithm 4 gets as inputs an admissible DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ over $\mathcal{L}$, a set of leading (minimal) diagnoses $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ such that $|\mathbf{D}| \geq 2$ and a parameter $q \in \mathbb{N} \cup \{\infty\}, q \geq 1$ that indicates the number of queries in $\mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ the algorithm is supposed to return (where $q := \infty$ signalizes that a maximum number of queries should be output). The way of generating a pool of queries is guided by Proposition 7.4 which says that a non-empty set Q of formulas over $\mathcal{L}$ is a query w.r.t. $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ if and only if $\mathbf{D}^+(Q)$ as well as $\mathbf{D}^-(Q)$ are non-empty sets of diagnoses. That is, the necessary and sufficient criteria for Q to be a query are

---

Algorithm 4 Generation of Queries and Q-Partitions

Input: an admissible DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, a set of minimal diagnoses $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ such that $|\mathbf{D}| \geq 2$, a desired number $q \in \mathbb{N} \cup \{\infty\}$, $q \geq 1$ of queries w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ to be returned

Output: a set $\mathbf{QP}$ including tuples $\langle Q, \langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q) \rangle \rangle$ such that: If $q \geq |\mathbf{QP}_{\max}|$, then

1. there are no two tuples $\langle Q, \mathfrak{P}(Q) \rangle$, $\langle Q', \mathfrak{P}(Q') \rangle$ in $\mathbf{QP}$ such that $Q = Q'$ *or* $\mathfrak{P}(Q) = \mathfrak{P}(Q')$, and

2. $\mathbf{QP}$ includes a tuple $\langle Q, \langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q) \rangle \rangle$ only if $Q \in \mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, and

3. $\mathbf{QP}$ includes at most one tuple where $\mathbf{D}^+(Q) = Y$ for each $Y \subset \mathbf{D}$, and

4. for each $Y \subset \mathbf{D}$ for which a query Q w.r.t. $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ exists such that (a) Q includes only entailments computed by the used GETENTAILMENTS function and (b) $\mathfrak{P}(Q)$ is such that $\mathbf{D}^+(Q) = Y$, $\mathbf{QP}$ includes a tuple $\langle Q', \mathfrak{P}(Q') \rangle$ such that $\mathbf{D}^+(Q') = Y$, and

5. $\mathbf{QP} \neq \emptyset$.

If $q < |\mathbf{QP}_{\max}|$, then $\mathbf{QP}$ includes q tuples satisfying (1), (2) and (3). ($|\mathbf{QP}_{\max}| \geq 0$ is the maximum number of tuples $\langle Q, \mathfrak{P}(Q) \rangle$ that can be computed by GETPOOLOFQUERIES by the used GETENTAILMENTS function)

1: **procedure** GETPOOLOFQUERIES($\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R, \mathbf{D}, q$)
2:     $E_{\mathbf{D}} \leftarrow \emptyset$
3:     **for** $\mathcal{D} \in \mathbf{D}$ **do**
4:         $E_{\mathcal{D}} \leftarrow$ GETENTAILMENTS($\mathcal{D}, \mathcal{K}, \mathcal{B}, P$) $\triangleright E_{\mathcal{D}_r}$ is the set of entailments of $\mathcal{K}_r^*$
5:         $E_{\mathbf{D}} \leftarrow E_{\mathbf{D}} \cup \{\langle \mathcal{D}, E_{\mathcal{D}} \rangle\}$
6:     **for** $\emptyset \subset \mathbf{S} \subset \mathbf{D}$ **do**
7:         $isQuery \leftarrow false$
8:         $Q \leftarrow$ GETCOMMONENTAILMENTS($\mathbf{S}, E_{\mathbf{D}}$)
9:         **if** $Q \neq \emptyset$ **then**
10:             **for** $\mathcal{D}_r \in \mathbf{D} \setminus \mathbf{S}$ **do**
11:                 **if** $Q \subseteq E_{\mathcal{D}_r}$ **then** $\triangleright$ Does $\mathcal{K}_r^* \models Q$?
12:                     $\mathbf{D}^+ \leftarrow \mathbf{D}^+ \cup \{\mathcal{D}_r\}$
13:                 **else if** $\neg$ISKBVALID($\mathcal{K}_r^* \cup Q, \langle \cdot, \emptyset, \emptyset, N \rangle_R$) **then** $\triangleright$ ISKBVALID (see Algorithm 1)
14:                     $\mathbf{D}^- \leftarrow \mathbf{D}^- \cup \{\mathcal{D}_r\}$
15:                     $isQuery \leftarrow true$
16:                 **else**
17:                     $\mathbf{D}^0 \leftarrow \mathbf{D}^0 \cup \{\mathcal{D}_r\}$
18:             **if** $isQuery \wedge \neg$INCLQPART($\mathbf{QP}, \langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle$) **then**
19:                 $Q' \leftarrow$ MINQ($\emptyset, Q, \emptyset, \langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$)
20:                 $\mathbf{QP} \leftarrow \mathbf{QP} \cup \{\langle Q', \langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle \rangle\}$
21:                 **if** $|\mathbf{QP}| = q$ **then**
22:                     **return** $\mathbf{QP}$
23:     **if** $|\mathbf{QP}| = 0$ **then**
24:         $\mathbf{QP} \leftarrow$ ADDTRIVIALQUERIES($\mathbf{D}, \mathbf{QP}$)
25:     **return** $\mathbf{QP}$

26: **procedure** MINQ($X, Q, QB, \langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$)
27:     **if** $X \neq \emptyset \wedge$ ISQPARTCONST($QB, \langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$) **then**
28:         **return** $\emptyset$
29:     **if** $|Q| = 1$ **then**
30:         **return** Q
31:     $k \leftarrow$ SPLIT($|Q|$)
32:     $Q_1 \leftarrow$ GET($Q, 1, k$)
33:     $Q_2 \leftarrow$ GET($Q, k+1, |Q|$)
34:     $Q_2^{\min} \leftarrow$ MINQ($Q_1, Q_2, QB \cup Q_1, \langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$)
35:     $Q_1^{\min} \leftarrow$ MINQ($Q_2^{\min}, Q_1, QB \cup Q_2^{\min}, \langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$)
36:     **return** $Q_1^{\min} \cup Q_2^{\min}$

37: **procedure** ISQPARTCONST($Q, \langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$)
38:     **for** $\mathcal{D}_r \in \mathbf{D}^-$ **do**
39:         **if** ISKBVALID($\mathcal{K}_r^* \cup Q, \langle \cdot, \emptyset, \emptyset, N \rangle_R$) **then** $\triangleright$ ISKBVALID (see Algorithm 1)
40:             **return** $false$
41:     **for** $\mathcal{D}_r \in \mathbf{D}^0$ **do**
42:         **if** $\mathcal{K}_r^* \models Q$ **then**
43:             **return** $false$
44:     **return** $true$

	$E_{\mathcal{D}_3}$	$E_{\mathcal{D}_4}$
e_1	$\forall X a_1(X) \to a_2(X)$	$\forall X a_1(X) \to a_2(X)$
e_2	$\forall X a_1(X) \to m_1(X)$	$\forall X a_1(X) \to m_1(X)$
e_3	$\forall X a_1(X) \to m_2(X)$	$\forall X a_1(X) \to m_2(X)$
e_4	$\forall X m_1(X) \to b(X)$	$\forall X m_1(X) \to b(X)$
e_5	$\forall X a_1(X) \to b(X)$	$\forall X a_1(X) \to b(X)$
e_6	$a_2(w)$	$a_2(w)$
e_7	$a_2(u)$	$a_2(u)$
e_8	$m_1(w)$	$m_1(w)$
e_9	$m_1(u)$	$m_1(u)$
e_{10}	$m_2(w)$	$m_2(w)$
e_{11}	$m_2(u)$	$m_2(u)$
e_{12}	$b(w)$	$b(w)$
e_{13}	$b(u)$	$b(u)$
e_{14}		$\forall X b(X) \to m_3(X)$
e_{15}		$\forall X c(X) \to m_3(X)$
e_{16}		$\forall X m_1(X) \to m_3(X)$
e_{17}		$\forall X a_1(X) \to m_3(X)$
e_{18}		$m_3(w)$
e_{19}		$m_3(u)$

Table 8.2: (Example 8.1) Entailments computed for KBs $\mathcal{K}_3^*$ and $\mathcal{K}_4^*$.

(CQ1) $Q \neq \emptyset$ and

(CQ2) $\mathbf{D}^+(Q) \neq \emptyset$ and

(CQ3) $\mathbf{D}^-(Q) \neq \emptyset$.

Note, since the disjoint sets of diagnoses $\mathbf{D}^+(Q) \subseteq \mathbf{D}$ and $\mathbf{D}^-(Q) \subseteq \mathbf{D}$ must not be empty, $|\mathbf{D}| \geq 2$ must be postulated in order for any queries to exist w.r.t. $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ (cf. Corollary 7.4).

As a first action (lines 3-5), the algorithm computes a set of entailments $E_{\mathcal{D}_i}$ for each $\mathcal{K}_i^*$ (cf. Formula 7.1) where $\mathcal{D}_i \in \mathbf{D}$ and stores these entailments along with the respective diagnosis as a tuple $\langle \mathcal{D}_i, E_{\mathcal{D}_i} \rangle$ in a set $E_{\mathbf{D}}$. This is accomplished by the function GETENTAILMENTS which gets a tuple $\langle X, Y, Z, W \rangle$ of arguments where X, Y, Z are sets of formulas over some logic $\mathcal{L}$ and W is a set including sets of formulas over $\mathcal{L}$. Then, GETENTAILMENTS computes a *finite* (cf. Remark 2.3) set of entailments of certain types (cf. Examples 8.1 and 8.6) of the KB $(Y \setminus X) \cup Z \cup U_W$.

Then, the algorithm runs through all proper non-empty subsets $\mathbf{S}$ of the leading diagnoses $\mathbf{D}$ and, for each $\mathbf{S}$, it computes the set of common entailments Q of all KBs $\mathcal{K}_i^*$ where $\mathcal{D}_i \in \mathbf{S}$ (function GET-COMMONENTAILMENTS) by means of the precomputed set $E_{\mathbf{D}}$. That is, $Q := \bigcap_{\mathcal{D} \in \mathbf{S}} E_{\mathcal{D}}$. If Q is non-empty, then CQ1 and CQ2 are fulfilled for Q. CQ2 is met since $\mathbf{S} \neq \emptyset$ and thus there is a diagnosis $\mathcal{D}_i \in \mathbf{D}$ such that $\mathcal{K}_i^* \models Q$ which implies that $\mathbf{D}^+(Q) \neq \emptyset$. So, the algorithm proceeds to verify CQ3 (lines 10-17) in that it assigns the remaining diagnoses in $\mathbf{D}$ that are not in $\mathbf{S}$ to the according sets $\mathbf{D}^+(Q)$, $\mathbf{D}^-(Q)$ or $\mathbf{D}^0(Q)$ as per Definition 7.2. Note that the function ISKBVALID has been speci-

fied in Algorithm 1 on page 48. With the parameters given when called in line 13, ISKBVALID checks whether $\mathcal{K}_r^* \cup Q = (\mathcal{K} \setminus \mathcal{D}_r) \cup \mathcal{B} \cup U_{P \cup \{Q\}}$ does not violate any requirement in R and does not entail any test case in N. Once the call to this function returns *false* for one diagnosis $\mathcal{D}_r \in \mathbf{D} \setminus \mathbf{S}$, it holds that $\mathcal{D}_r \in \mathbf{D}^-(Q)$ thus CQ3 is definitely met. Therefore, *isQuery* is set to *true* in line 15. If, on the other hand, *isQuery* is not set to *true* for any diagnosis in $\mathbf{D} \setminus \mathbf{S}$, then the set $\mathbf{D}^-(Q) = \emptyset$ and thus Q is not in $\mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$.

So far, we have proven the following proposition.

Proposition 8.1. *Let a DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, *a set of diagnoses* $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ *and a natural number* $q \geq 1$ *be the input to the function* GETPOOLOFQUERIES. *Then, a value stored in variable* Q *at the time* GETPOOLOFQUERIES *executes line 18 is a query w.r.t.* $\mathbf{D}$ *and* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *iff the variable* isQuery *stores the value* true.

If the purpose was only to find queries (and not q-partitions), the algorithm could stop processing for the current Q and go to the next set $\mathbf{S}$, given that *isQuery* is set to *true* for some diagnosis. However, as the q-partition provides meaningful information to assess a query, e.g. it gives the number of diagnoses invalidated for each answer or the estimated probability of each answer (cf. Chapter 7), the q-partition is a necessary input to the subsequently called function SELECTBESTQUERY (line 48 in Algorithm 6, see later in Sections 9.2.4 and 9.3) that selects a query from the pool of queries $\mathbf{QP}$. For this reason, the algorithm continues until the computation of the q-partition for Q is complete.

In a last step (lines 18-20), given that *isQuery* is *true* and there is not yet a query with the same q-partition in $\mathbf{QP}$, the algorithm computes a set-minimal subset $Q_{\min}$ of Q such that the q-partition of $Q_{\min}$ is the same as the one of Q (function MINQ). Finally, the tuple $\langle Q_{\min}, \langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle \rangle$ including the minimized query $Q_{\min}$ along with its q-partition $\langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle$ is added to $\mathbf{QP}$. If $|\mathbf{QP}| = q$, then $\mathbf{QP}$ is returned; otherwise, a further iteration for another $\mathbf{S}$ is executed. If $|\mathbf{QP}| = q$ is not met until all seeds $\mathbf{S}$ have been processed, the set $\mathbf{QP}$ is checked for emptiness in line 23. If $\mathbf{QP} = \emptyset$, then the function ADDTRIVIALQUERIES (line 24) adds $|\mathbf{D}| \geq 2$ queries as defined by Q in Proposition 7.5 to $\mathbf{QP}$ (cf. Corollary 7.4) and then returns $\mathbf{QP}$; otherwise, $\mathbf{QP}$ is directly returned.

Remark 8.1 Notice that lines 23 and 24 in Algorithm 4 aim at ensuring the non-emptiness of the pool of queries $\mathbf{QP}$ returned by GETPOOLOFQUERIES for *any* GETENTAILMENTS function (see Example 8.6 for different specifications of the GETENTAILMENTS function). This is a necessary criterion for the interactive KB debugging system (Algorithm 5) to work in a sound way since it guarantees that the CALCQUERY function (line 16 in Algorithm 5) *always* returns a query w.r.t. the current set of leading diagnoses $\mathbf{D}$ and the given DPI. Note that the $|\mathbf{D}|$ queries generated and added to $\mathbf{QP}$ by ADDTRIVIALQUERIES can be *trivially* obtained without the consultation of a reasoning service by extraction of the respective formulas from the KB $\mathcal{K}$, as prescribed by Proposition 7.5. □

8.2 Discussion of Query Pool Generation

Multiple Equal Q-Partitions. In the general case there is more than one query w.r.t. one and the same q-partition. For that reason alone that a minimized query is a set-minimal subset of an initially computed one where multiple such subsets may exist.

Example 8.2 An example for such a query resulting in multiple minimized subqueries with identical q-partition can be found in Example 8.1. □

However, note that GETPOOLOFQUERIES is designed to compute a pool $\mathbf{QP}$ that includes at most one query with one and the same q-partition. The idea behind this is (1) to minimize the calls to the

expensive function MINQ and (2) that two queries with the same q-partition have exactly the same properties w.r.t. common query selection criteria such as maximum expected information gain or maximum worst case invalidation rate of diagnoses after the query answer is known. Such criteria have been shown to often lead to a reduction of debugging effort for the interacting user (cf. [SFFR12, RSFF13]). As the purpose of the computation of the pool of queries **QP** is to constitute an input to the query selection function that uses exactly such selection measures, the inclusion of only one query with a particular q-partition is reasonable, also (3) to minimize computation time of the query selection function which needs to go through all elements of **QP** in order to pick the "best" one in the worst case.

On the other hand, regarding the comprehensibility of the query, i.e. the cognitive load on the user when it comes to understanding the meaning of the query, two queries with the same q-partition may well be significantly different. This however is beyond the scope of this work and considered a topic for future research.

The following proposition gives evidence that the set **QP** returned by GETPOOLOFQUERIES is indeed duplicate-free w.r.t. the q-partitions in **QP**.

Proposition 8.2. *Let a DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, *a set of diagnoses* $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ *and* $q \in \mathbb{N} \cup \{\infty\}, q \geq 1$ *be the input to the function* GETPOOLOFQUERIES. *Then, the function* GETPOOLOF-QUERIES *returns a set* **QP** *including tuples of the form* $\langle Q, \mathfrak{P}(Q) \rangle$ *where* $Q \in \mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ *is a query and* $\mathfrak{P}(Q) = \langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q) \rangle$ *is the q-partition of* Q *such that* **QP** *does not include any two equal queries and does not include any two equal q-partitions.*

Proof. The test of the criterion ¬INCLQPART tested before the call to MINQ will always return false for the q-partition $\langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle$ if $\langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle$ is already included in a tuple in **QP**. Since MINQ is q-partition-preserving, no q-partition that does not occur in a tuple in **QP** can become equal to some q-partition in **QP** by a call to MINQ. Therefore, **QP** cannot include any two equal q-partitions. Since two equal queries have equal q-partitions, any two different q-partitions cannot be q-partitions of equal queries. Thus, **QP** cannot include any two equal queries either. □

Note that, on account of the q-partition preserving property of MINQ, only such q-partitions are ruled out by the criterion in line 18 that would lead to duplicates at the time they should be added to **QP** in line 20.

Computation of Entailments.   Generally, the (*theoretical*) number of entailments of a set of formulas is not finite. However, the entailments (of a certain type) returned by a reasoner are finite. For instance, asked for entailments of $\{A \sqsubseteq B \sqcap C\}$, a reasoner performing the classification reasoning service would give back $A \sqsubseteq B$ and $A \sqsubseteq C$, but not entailments like $A \sqsubseteq B \sqcup C$ or $A \sqsubseteq C \sqcap C \sqcap C$. That is, when we speak of entailments, then we mean entailments in the *practical* sense (cf. Remark 2.3), i.e. w.r.t. a reasoning service such as classification for DL KBs which computes all and only subsumptions $X \sqsubseteq Y$ such that Y is the most specific concept that subsumes X, or forward-chaining for Datalog KBs which computes all and only atoms that are entailed by the KB.

Example 8.3   If we recall Example 8.1, we see that the number of computed entailments of $\mathcal{K}_4^*$ and $\mathcal{K}_3^*$ was 19 and 13 respectively, which are rather high numbers in the light of the small KBs, but importantly these numbers are necessarily finite. For, there cannot be more than $|Pred|^2$ entailments of the $\forall X p_1(X) \rightarrow p_2(X)$ type and not more than $|Pred| \, |Const|$ entailments of the $p(a)$ type for a KB whose signature includes the unary predicate symbols $Pred$ and constant symbols $Const$ and does not include any function symbols. In case of KB $\mathcal{K}_3^*$, for example, the set $Pred = \{a_1, a_2, m_1, m_2, m_3, a, b\}$ and $Const = \{u, w\}$ which means that upper bounds for the number of entailments of the first and second type are 49 and 14, respectively. □

Further, note that the number of existing different q-partitions and which q-partitions there are at all w.r.t. some set of leading diagnoses $\mathbf{D}$ and a DPI depends on the function GETENTAILMENTS, i.e. on the set of entailments calculated by it.

Example 8.4 Recall Example 8.1 where we constructed a query Q w.r.t. the set of all minimal diagnoses for the DPI given by Table 15.2. Assume now that only entailments of the first type, i.e. those of the form $\forall X \, p_1(X) \rightarrow p_2(X)$, and none of the second type $p(a)$ are computed by GETENTAILMENTS and denote the set of entailments of this form of $\mathcal{K}_i^*$ by $E'_{\mathcal{D}_i}$. Then, $Q' = E'_{\mathcal{D}_3} \cup E'_{\mathcal{D}_4} = \{e_1, \ldots, e_5\}$ (cf. Table 8.2), i.e. a subset of the query Q computed for a GETENTAILMENTS function producing entailments of both types. The q-partition of Q' is the same as the q-partition of Q, namely $\langle \{\mathcal{D}_3, \mathcal{D}_4\}, \{\mathcal{D}_1, \mathcal{D}_2\}, \emptyset \rangle$. However, the queries $Q_{\min,1}$ and $Q_{\min,2}$ are no longer obtained as minimized versions of Q', unlike $Q_{\min,3}$ and $Q_{\min,4}$ which are subqueries of Q', too. □

Minimizing the Set $\mathbf{D}^0$ in Q-Partitions. Recall that $\mathbf{D}^0 = \emptyset$ is a desirable property of a q-partition since a query with such q-partition may invalidate any leading diagnosis, depending on the answer to the query (cf. Chapter 7). In other words, no leading diagnosis is guaranteed to be still valid *for any answer* after the query is added as a test case to the DPI.

In general, GETPOOLOFQUERIES computes q-partitions where $\mathbf{D}^0$ may be a non-empty set. However, if the GETENTAILMENTS function is specified to compute certain explicit entailments of $\mathcal{K}$, then $\mathbf{D}^0 = \emptyset$ can be guaranteed.

Definition 8.1 (Explicit Entailment). *Let $\mathcal{K}$ be a KB. Then, α is an explicit entailment of $\mathcal{K}$ iff $\alpha \in \mathcal{K}$.*

Now, if each set of entailments $E_\mathcal{D}$ computed by GETENTAILMENTS includes all the formulas that occur in some diagnosis in $\mathbf{D}$, but do not occur in $\mathcal{D}$, then GETPOOLOFQUERIES definitely returns a set $\mathbf{QP}$ of queries and associated q-partitions where $\mathbf{D}^0(Q) = \emptyset$ holds for each tuple in $\mathbf{QP}$.

Proposition 8.3. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI and $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. If the set $E_\mathcal{D}$ computed by* GETENTAILMENTS *meets $E_\mathcal{D} \supseteq U_\mathbf{D} \setminus \mathcal{D}$ for all $\mathcal{D} \in \mathbf{D}$, then* GETPOOLOFQUERIES *computes only queries Q with $\mathbf{D}^0(Q) = \emptyset$.*

Proof. Assume that Q is some query computed by GETPOOLOFQUERIES. As MINQ is a q-partition preserving transformation of Q, we can assume w.l.o.g. that Q is a query computed by GETPOOLOF-QUERIES *before* MINQ is called for Q. We have to show that for an arbitrary diagnosis $\mathcal{D}_i \in \mathbf{D}$ either $\mathcal{D}_i$ is assigned to $\mathbf{D}^+(Q)$ or to $\mathbf{D}^-(Q)$.

So, let us assume that there is a diagnosis $\mathcal{D}_k$ which is assigned to $\mathbf{D}^0(Q) = \mathbf{D} \setminus (\mathbf{D}^+(Q) \cup \mathbf{D}^-(Q))$ in line 17. Then, $Q \not\subseteq E_{\mathcal{D}_k}$ and $\mathcal{K}_k^* \cup Q$ does not violate any $x \in R \cup N$ must hold, otherwise $\mathcal{D}_k$ would have already been assigned to $\mathbf{D}^+(Q)$ in line 12 or to $\mathbf{D}^-(Q)$ in line 14. But $Q \not\subseteq E_{\mathcal{D}_k}$ implies $Q \not\subseteq U_\mathbf{D} \setminus \mathcal{D}_k$ since $E_{\mathcal{D}_k} \supseteq U_\mathbf{D} \setminus \mathcal{D}_k$ by precondition. This in turn means that there is some formula ax in Q which is not in $U_\mathbf{D} \setminus \mathcal{D}_k$. Then $ax \in \mathcal{D}_k$ must hold, as otherwise for all formulas $ax' \in Q$ it would hold that ax' is an entailment of $\mathcal{K}_k^* = (\mathcal{K} \setminus \mathcal{D}_k) \cup \mathcal{B} \cup U_P$, i.e. an entailment of all formulas in $\mathcal{K} \cup \mathcal{B} \cup U_P$ except for those in $\mathcal{D}_k$. However, all entailments of $\mathcal{K}_k^*$ are stored in $E_{\mathcal{D}_k}$ by the implementation of the function GETENTAILMENTS. Thus $Q \subseteq E_{\mathcal{D}_k}$ would hold which cannot be the case as shown before. Consequently, we have derived that $Q \cap \mathcal{D}_k \neq \emptyset$ which means by set-minimality of diagnoses in $\mathbf{D}$, in particular of $\mathcal{D}_k$, that $\mathcal{K}_k^* \cup Q$ must violate some $x \in R \cup N$ which is a contradiction to the assumption that $\mathcal{D}_k \in \mathbf{D}^0(Q)$. □

Example 8.5 Let us come back to the example DPI given by Table 15.2. The possibility of a query Q constructed by Algorithm 4 with $\mathbf{D}^0(Q) \neq \emptyset$ is witnessed by the selection of seed $\mathbf{S} = \{\mathcal{D}_1\}$ and the assumption that entailments of the two types given in Example 8.1 are produced by GETENTAILMENTS.

The set of entailments $Q = E_{\mathcal{D}_1} = \{e_4, e_{14}, e_{15}, \forall X m_2(X) \to d(X)\}$ (for e_i cf. Table 8.2). Then, $\mathcal{D}_2$ as well as $\mathcal{D}_3$ are assigned to $\mathbf{D}^-(Q)$ as both KBs $\mathcal{K}_3^* \cup Q$, $\mathcal{K}_4^* \cup Q$ entail $m_3(w)$ and $\neg m_3(w)$ wherefore they are both inconsistent and thus violate $r_1 \in R$. However, $\mathcal{D}_4 \in \mathbf{D}^0(Q)$ since $\mathcal{K}_i^* \not\models \forall X m_2(X) \to d(X)$ and hence does not entail Q and since $\mathcal{K}_i^* \cup Q$ does not violate consistency or coherency (recall that the set of negative test cases is empty in the DPI and thus must not be considered), i.e. does not contain a conflict set.

Applying Proposition 8.3, we could use a modified GETENTAILMENTS function that returns a minimal set of entailments just that the precondition of the proposition is met, i.e. $E'_{\mathcal{D}} = U_{\mathbf{D}} \setminus \mathcal{D}$ for all $\mathcal{D} \in \mathbf{D}$. With this function, for the seed $\mathbf{S} = \{\mathcal{D}_1\}$ we would get $Q' = E'_{\mathcal{D}_1} = \{2, 3, 4, 5\}$ (again, formulas in Table 15.2 are referred to just by their number). Let us now check whether $\mathbf{D}^0(Q')$ is indeed empty. As explicit entailments are stronger than non-explicit ones, we must still have that $\mathcal{D}_2, \mathcal{D}_3 \in \mathbf{D}^-(Q')$. For $\mathcal{D}_4$, we have $\mathcal{K}_4^* \cup Q' = \{1, 3, 5, 6, 7, 8\} \cup \{2, 3, 4, 5\} = \{1, 2, 3, 4, 5, 6, 7, 8\}$ which corresponds to the entire KB plus background knowledge of the given DPI and includes conflict sets $\mathcal{C}_1 = \{1, 3, 4\}$ and $\mathcal{C}_2 = \{1, 2, 3, 5\}$ wherefore it is inconsistent. Therefore, diagnosis $\mathcal{D}_4$ must also be an element of $\mathbf{D}^-(Q')$.

Please note that making the entailments $Q = E_{\mathcal{D}_1}$ computed by the unmodified GETENTAILMENTS function only slightly stronger would already suffice to force inclusion of $\mathcal{D}_4$ in $\mathbf{D}^0(Q)$. In fact, including $ax_4 := \forall X m_2(X) \to (\forall Y s(X, Y) \to a(Y)) \wedge d(X)$ in Q instead of $\forall X m_2(X) \to d(X)$ would make Q non-disjoint with $\mathcal{D}_4$ as both comprise ax_4. Consequently, in line with the proof of Proposition 8.3, $\mathcal{K}_4^* \cup Q$ must include a conflict set ($\{1, 3, 4\}$) wherefore $\mathcal{D}_4 \in \mathbf{D}^-(Q)$.

Another point we want to mention is that empty $\mathbf{D}^0$ *could* also be achieved by making the query slightly weaker. For our concrete query $Q = E_{\mathcal{D}_1}$, this means that leaving out $\forall X m_2(X) \to d(X)$ would lead to empty $\mathbf{D}^0(Q)$. However, the difference to the scenario above where we made Q sightly stronger is that $\mathcal{D}_4$ would be an element of $\mathbf{D}^+(Q)$ instead of $\mathbf{D}^-(Q)$ in this case, i.e. the q-partition would be $\langle \{\mathcal{D}_1, \mathcal{D}_4\}, \{\mathcal{D}_2, \mathcal{D}_3\}, \emptyset \rangle$.

A shortcoming of the strategy of making the query weaker is that it can be computationally expensive as perhaps a large number of subsets of Q might need to be considered and tested for fulfillment of $\mathbf{D}^0(Q) = \emptyset$. Each such test would involve calls to the reasoner which are usually expensive. A second drawback is that no guarantee is given to finally end up with an empty set $\mathbf{D}^0(Q)$ since weakening of Q might also involve the "shift" of some diagnosis from $\mathbf{D}^-(Q)$ to $\mathbf{D}^0(Q)$. On the other hand, the strategy of computing stronger entailments is computationally more resource-saving as (trivially obtained) explicit entailments can be added to make the query stronger. Furthermore, making the query stronger – in a controlled way, by adding formulas from $U_{\mathbf{D}} \setminus U_{\mathbf{D}^+(Q)}$ to Q as suggested by Proposition 8.3 – can never lead to non-empty $\mathbf{D}^0(Q)$ as Proposition 8.3 substantiates. $\square$

(Non-)Completeness of Query Pool QP.   Note that specifying $q := \infty$ causes GETPOOLOFQUERIES to run through all $\mathbf{S} \subset \mathbf{D}$ and to compute a maximum number of queries. However, in general, not all theoretically possible queries are computed by GETPOOLOFQUERIES. One trivial reason for this is that only minimized, i.e. set-minimal, queries are contained in the returned set $\mathbf{QP}$.

But, also queries Q' with $\mathbf{D}^+(Q') = Y \subset \mathbf{D}$ will not be included in $\mathbf{QP}$ if there is some query Q with $\mathbf{D}^+(Q) = Y$ such that $|\mathbf{D}^-(Q)| > |\mathbf{D}^-(Q')|$ (and, equivalently, $|\mathbf{D}^0(Q)| < |\mathbf{D}^0(Q')|$). As we will learn in a moment, both mentioned reasons for the incompleteness of the output of GETPOOLOFQUERIES will even be desirable for reasons of efficiency. That is, the mentioned types of queries that are not taken into account in $\mathbf{QP}$ are "non-preferred" as non-set-minimal queries demand a non-necessary amount of user interaction and the answering of queries Q with a non-necessarily large set $\mathbf{D}^0(Q)$ involves a worse discrimination between leading minimal diagnoses (and, if these are "good" representatives of all minimal diagnoses, then of all minimal diagnoses) than other queries Q' with $|\mathbf{D}^0(Q')| < |\mathbf{D}^0(Q)|$ and $\mathbf{D}^+(Q) = \mathbf{D}^+(Q')$.

Still, GETPOOLOFQUERIES meets a completeness criterion for a subset of all queries $\mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, elements of which cannot be *trivially* detected to be "non-preferred". That is, GETPOOLOFQUERIES is complete w.r.t. the set $\mathbf{D}^+$, as the following proposition states. In other words, for each subset $X \subset \mathbf{D}$ it detects a q-partition with $\mathbf{D}^+ = X$, if one exists.

Proposition 8.4. *Let a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ such that $|\mathbf{D}| \geq 2$ and some $q \in \mathbb{N} \cup \{\infty\}, q \geq 1$ be the inputs to GETPOOLOFQUERIES and let $|\mathbf{QP}_{\max}| \geq 0$ be the maximum number of tuples $\langle Q, \mathfrak{P}(Q) \rangle$ that can be computed by GETPOOLOFQUERIES by means of the used GETENTAILMENTS function. Further, let Y be an arbitrary subset of $\mathbf{D}$. If there is some query $Q \in \mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ that (1) includes only entailments that are computed by GETENTAILMENTS and (2) has a q-partition such that $\mathbf{D}^+(Q) = Y$, then GETPOOLOFQUERIES with parameter $q \geq |\mathbf{QP}_{\max}|$ returns a set $\mathbf{QP}$ including a query Q' with $\mathbf{D}^+(Q') = Y$. Moreover, this query Q' is found in the iteration where the seed $\mathbf{S} = Y$.*

Proof. Since $q \geq |\mathbf{QP}_{\max}|$, GETPOOLOFQUERIES will arrive at a step where it selects the seed $\mathbf{S} = Y$ in line 6. Now, let us assume that in this iteration no query Q with $\mathbf{D}^+(Q) = Y$ is found. Then, either (a) no query is found at all, i.e. CQ1 or CQ2 or CQ3 are violated, or (b) a query Q with $\mathbf{D}^+(Q) \neq Y$ is found.

(a): Assume first that CQ1 is violated, i.e. GETCOMMONENTAILMENTS called with argument $\mathbf{S}$ returns $\emptyset$. This implies that the KBs $\mathcal{K}_r^*$ for $\mathcal{D}_r \in Y$ have no common entailments, if entailments are computed by GETENTAILMENTS. This however means that there cannot be a q-partition with $\mathbf{D}^+ \supseteq Y$ which is a contradiction to the precondition that there is some query $Q \in \mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ that includes only entailments computed by GETENTAILMENTS and has a q-partition such that $\mathbf{D}^+(Q) = Y$.

Second, assume that CQ2 is violated, i.e. $\mathbf{D}^+(Q) = \emptyset$. If GETCOMMONENTAILMENTS with argument $\mathbf{S}$ returned $Q \neq \emptyset$, then $\mathbf{D}^+(Q) \supseteq \mathbf{S} \supset \emptyset$ would hold. Thus, $Q = \emptyset$, i.e. CQ1 is violated. So, as shown before, this leads to a contradiction.

In case any of CQ1 or CQ2 is violated, we already derived a contradiction. So, we make the assumption that CQ1 and CQ2 are met. So, finally, let us assume that CQ3 is violated, i.e. that $\mathbf{D}^-(Q) = \emptyset$. That is, if Q (which must be a non-empty set by CQ1) denotes *all* common entailments (computable with GETENTAILMENTS) of $\mathcal{K}_r^*$ for $\mathcal{D}_r \in Y$, then $\mathcal{K}_i^* \cup Q$ does not violate any $x \in R \cup N$ for any $\mathcal{D}_i \in \mathbf{D} \setminus \mathbf{S}$. Consequently, for all diagnoses $\mathcal{D}_i$ in $\mathbf{D}$ we have that $\mathcal{K}_i^* \cup Q$ does not violate any $x \in R \cup N$. But, as there is, by precondition, a query with $\mathbf{D}^+ = Y$, this query must be a subset of all possible common entailments (computable with GETENTAILMENTS) of KBs $\mathcal{K}_i^*$ for diagnoses in Y, i.e. this query must be a subset of Q. But, by monotonicity of $\mathcal{L}$, no $\mathcal{K}_i^* \cup Q'$ for a subset Q' of Q can violate $x \in R \cup N$ if Q does not. Again, we have a contradiction to the precondition as above.

(b): Here, a query Q is found with $\mathbf{D}^+(Q) \neq Y$ and $\mathbf{D}^-(Q) \neq \emptyset$. Since Q is a query, $Q \neq \emptyset$ must hold. Since the seed $\mathbf{S} = Y$, this means that Q is the set of all common entailments (computable with GETENTAILMENTS) of $\mathcal{K}_i^*$ for $\mathcal{D}_i \in Y$, i.e. $\mathbf{D}^+(Q) \supseteq Y$. By $\mathbf{D}^+(Q) \neq Y$, we conclude that $\mathbf{D}^+(Q) \supset Y$ must be true. The only way of achieving a smaller set $\mathbf{D}^+(Q)$, namely $\mathbf{D}^+(Q) = Y$, is to add some formulas to Q as making Q smaller can only increase $\mathbf{D}^+(Q)$. This holds because postulating that, instead of Q, only a subset Q' of Q must be entailed by $\mathcal{K}_i^*$, can cause a new KB $\mathcal{K}_j^*$ for diagnosis $\mathcal{D}_j \notin \mathbf{D}^+(Q)$ to entail Q'. However, as Q is the set of *all* entailments computable with GETENTAILMENTS of KBs $\mathcal{K}_i^*$ for $\mathcal{D}_i \in Y$, a superset Q'' of Q computed by GETENTAILMENTS with $\mathbf{D}^+(Q'') = Y$ can never be obtained. Therefore, we have a contradiction to the precondition.

We have now proven the following: If there exists a q-partition as described in the proposition, then this q-partition is found in the iteration where the seed $\mathbf{S} = Y$. $\qquad \square$

Remark 8.2 Regarding Proposition 8.4, note the following:

(a) In fact, as one and the same q-partition must occur at most once in $\mathbf{QP}$, GETPOOLOFQUERIES must only keep assigning diagnoses in $\mathbf{D} \setminus \mathbf{S}$ to the respective sets of the q-partition as long as $\mathbf{D}^+ = \mathbf{S}$. Because for $\mathbf{D}^+ = Z \supset \mathbf{S}$, we know to find a query (if one exists) for the seed $\mathbf{S} = Z$.

(b) A statement equivalent to the proposition is: If there is no query (including only entailments computed by the GETENTAILMENTS function) with $\mathbf{D}^+ = Y$ found for seed $\mathbf{S} = Y$, then such a query and q-partition, respectively, does not exist. $\qquad\square$

The following proposition states that if a q-partition with one and the same set $\mathbf{D}^+$ is found twice during the execution of GETPOOLOFQUERIES, then the queries for both q-partitions and thus both q-partitions must be equal. That is, for one set $\mathbf{D}^+$, there is at most one tuple in $\mathbf{QP}$.

Proposition 8.5. *Let Q_i be a query with $\mathbf{D}^+(Q_i) = Y$ in the set $\mathbf{QP}$ returned by* GETPOOLOFQUERIES *and found for seed $\mathbf{S}_i = Y$ and let Q_j be a query with $\mathbf{D}^+(Q_j) = Y$ in the set $\mathbf{QP}$ returned by* GETPOOLOFQUERIES *and found for some seed $\mathbf{S}_j \subset Y$. Then $Q_i = Q_j$.*

Proof. Let Q_i', Q_j' be the queries stored in the variable Q in line 18 for seeds $\mathbf{S}_i$ and $\mathbf{S}_j$, respectively; i.e. the supersets of the queries Q_i, Q_j *before* the minimization function MINQ is called for each of them. $Q_j' \subseteq Q_i'$ holds by the fact that Q_i' is the set of *all* common entailments computable with GETENTAILMENTS of $\mathcal{K}_r^*$ for $\mathcal{D}_r \in Y$ and by the fact that Q_j' must be a set of common entailments computed by GETENTAILMENTS of exactly these KBs, because of $\mathbf{D}^+(Q_j') = Y$ and Definition 7.2. $Q_j' \supseteq Q_i'$ holds by the fact that Q_j' is computed as intersection of $E_{\mathcal{D}_r}$ where $\mathcal{D}_r \in \mathbf{S}_j$ and Q_i' is computed as intersection of $E_{\mathcal{D}_s}$ where $\mathcal{D}_s \in \mathbf{S}_i \supset \mathbf{S}_j$. Thus, we can conclude that $Q_i' = Q_j'$.

As $Q_i' = Q_j'$, also $\mathfrak{P}(Q_i') = \mathfrak{P}(Q_j')$ must hold for the q-partitions by Proposition 7.2. That the minimized versions Q_i, Q_j of Q_i', Q_j' output by MINQ are equal, follows from the determinism of the MINQ function, wherefore equal inputs, i.e. $(\emptyset, Q_i', \emptyset, \mathfrak{P}(Q_i'), \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R) = (\emptyset, Q_j', \emptyset, \mathfrak{P}(Q_j'), \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R)$, must yield equal outputs. $\qquad\square$

Remark 8.3 Proposition 8.5 hints at a possible improvement of Algorithm 4, namely to check in line 6 whether the seed $\mathbf{S}$ already occurs as a set $\mathbf{D}^+$ in some tuple in $\mathbf{QP}$ and only continue the execution for $\mathbf{S}$ if this does not hold (not shown in Algorithm 4). In this vein, time and reasoning costs (line 14) can be saved.

Another improvement regarding line 6 is to delete all remaining seeds $\mathbf{S}'$ with the property $\mathbf{S}' \supset \mathbf{S}$ if Q in line 8 is the empty set (not shown in Algorithm 4). Namely, all seeds $\mathbf{S}'$ must also lead to $Q = \emptyset$ since the intersection of $E_{\mathcal{D}}$ for $\mathcal{D} \in \mathbf{S}$ already returned $\emptyset$ wherefore the intersection of $E_{\mathcal{D}}$ for $\mathcal{D} \in \mathbf{S}'$ must also return $\emptyset$. $\qquad\square$

By now, we know from Proposition 8.5 that, given a query with $\mathbf{D}^+$ exists, one and only one q-partition with $\mathbf{D}^+$ will be added to $\mathbf{QP}$, but which one?

W.r.t. one and the same set $\mathbf{D}^+$, queries with a set $\mathbf{D}^-$ with higher cardinality are preferable over others as the cardinality of $\mathbf{D}^0$ should be minimized (cf. Chapter 7). So, preferable queries among those with equal set $\mathbf{D}^+$ are those for which $\mathbf{D}^-$ is a set-maximal set. Exactly such a query is added to $\mathbf{QP}$ for each $\mathbf{D}^+$ for which a query exists, as the following proposition shows.

Proposition 8.6. *If the set $\mathbf{QP}$ returned by* GETPOOLOFQUERIES *comprises a query Q with $\mathbf{D}^+(Q) = Y$, then Q is a query with minimal $|\mathbf{D}^0(Q)|$ among all queries Q' with $\mathbf{D}^+(Q') = Y$ computable with the function* GETENTAILMENTS.

Proof. Assume that GETPOOLOFQUERIES finds a query Q with $\mathbf{D}^+(Q) = Y$ and $|\mathbf{D}^0(Q)| = k$ and assume there is a query Q' (consisting only of entailments computed by function GETENTAILMENTS) with $\mathbf{D}^+(Q') = Y$ and with $|\mathbf{D}^0(Q')| < k$. This means that $|\mathbf{D}^-(Q)| < |\mathbf{D}^-(Q')|$. However, as Q is computed for seed $\mathbf{S} = Y$, Q is a maximal set of entailments computable with GETENTAILMENTS of $\mathcal{K}_i^*$ for $\mathcal{D}_i \in Y$. Because Q' is also a common entailment of $\mathcal{K}_i^*$ for $\mathcal{D}_i \in Y$, we have that $Q' \subseteq Q$ must be true. Since the fact that $\mathcal{K}_i^* \cup Q$ does not violate any $x \in R \cup N$, i.e. the fact that $\mathcal{D}_i \notin \mathbf{D}^-(Q)$, implies by monotonicity of $\mathcal{L}$ that $\mathcal{K}_i^* \cup Q'$ for the subset Q' of Q cannot violate any $x \in R \cup N$ either, i.e. $\mathcal{D}_i \notin \mathbf{D}^-(Q')$, we conclude that $|\mathbf{D}^-(Q')| \leq |\mathbf{D}^-(Q)|$ must hold. This is a contradiction. $\qquad\square$

8.3 Minimization of Queries

MINQ. The minimization of the query Q by MINQ (see Algorithm 4) while preserving the q-partition aims at simplifying the job of the answering user who only needs to go through a smaller set of logical formulas $Q_{\min}$ in order to come up with an answer to the query. Since the q-partition reflects the properties of a query w.r.t. the invalidation of (leading) diagnoses and two queries have equal such properties, then of course the one that is a subset of the other should be asked.

The concept of the function MINQ is similar to the one of QX (Algorithm 1). Like QX, MINQ carries out a divide-and-conquer strategy to find a set-minimal set with a monotonic property. In this case, the monotonic property is not the invalidity of a subset of the KB w.r.t. a DPI (as per Definition 3.3) as it is for the computation of minimal conflict sets using QX, but the property of some $Q_{\min} \subset Q$ having the same q-partition as Q. So, the crucial difference between QX and MINQ is the function that checks this monotonic property. For MINQ, this function – that checks a subset of a query for constant q-partition – is ISQPARTCONST.

MINQ – Input Parameters. MINQ gets five parameters as input. The first three, namely X, Q and QB, are relevant for the divide-and-conquer execution, whereas the last two, namely the original q-partition $\langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle$ of the query (i.e. the parameter Q) that should be minimized, and the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ are both needed as an input to the function ISQPARTCONST. Besides the latter two, another argument QB is passed to this function where QB is a subset of the original query Q. ISQPARTCONST then checks whether the q-partition for the (potential) query QB is equal to the q-partition $\langle \mathbf{D}^+, \mathbf{D}^-, \mathbf{D}^0 \rangle$ of the original query given as argument. The DPI is required as the parameters $\mathcal{K}, \mathcal{B}, P, N$ and R are necessary for these checks.

MINQ – Testing Sub-Queries for Constant Q-Partition. In particular, ISQPARTCONST tests for each $\mathcal{D}_r \in \mathbf{D}^-$ whether $\mathcal{K}_r^* \cup QB$ is valid (w.r.t. $\langle \cdot, \emptyset, \emptyset, N \rangle_R$). If so, this means that $\mathcal{D}_r \notin \mathbf{D}^-(QB)$ and thus that the q-partition of QB is different to the one of Q wherefore *false* is immediately returned. If *true* for all $\mathcal{D}_r \in \mathbf{D}^-$, it is tested for $\mathcal{D}_r \in \mathbf{D}^0$ whether $\mathcal{K}_r^* \models QB$. If so, this means that $\mathcal{D}_r \notin \mathbf{D}^0(QB)$ and thus that the q-partition of QB is different to the one of Q wherefore *false* is immediately returned. If *false* is not returned for any $\mathcal{D}_r \in \mathbf{D}^-$ or $\mathcal{D}_r \in \mathbf{D}^0$, then the conclusion is that QB is a query w.r.t. to $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and has the same q-partition as Q wherefore the function returns *true*.

Note that, instead of calling a reasoner to answer whether $\mathcal{K}_r^* \models QB$, the set of precalculated entailments $E_{\mathcal{D}_r}$ of $\mathcal{K}_r^*$ for each $\mathcal{D}_r \in \mathbf{D}$ can be given as an argument to MINQ as well as to ISQPARTCONST (not shown in Algorithm 4). In this case an equivalent test is $QB \subseteq E_{\mathcal{D}_r}$. Such a strategy is particularly appropriate if reasoning is expensive for the DPI at hand.

Soundness of ISQPARTCONST is proven by the following lemma.

Lemma 8.1. *Let* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *be a DPI,* $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, $Q \in \mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ *with q-partition* $\mathfrak{P}(Q) = \langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q) \rangle$. *Then a non-empty set* $QB \subset Q$ *is a query in* $\mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ *with* $\mathfrak{P}(QB) = \mathfrak{P}(Q)$ *if*

1. $\forall \mathcal{D}_r \in \mathbf{D}^-(Q) : \mathcal{K}_r^* \cup QB$ *violates some* $r \in R$ *or entails some* $n \in N$ *and*

2. $\forall \mathcal{D}_r \in \mathbf{D}^0(Q) : \mathcal{K}_r^* \not\models QB$.

Proof. Let $Q \in \mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ and QB be an arbitrary proper subset of Q. If criterion 1) of this lemma is met, then we know that each diagnosis in $\mathbf{D}^-(Q)$ is in $\mathbf{D}^-(QB)$ as well, i.e. (I): $\mathbf{D}^-(QB) \supseteq \mathbf{D}^-(Q)$ holds.

Assume a minimal diagnosis $\mathcal{D}_r \in \mathbf{D}^0(Q)$. Then, $\mathcal{K}_r^* \cup Q$ does not violate any $r \in R$ and does not entail any $n \in N$ and $\mathcal{K}_r^*$ does not entail Q. This however implies that $\mathcal{K}_r^* \cup QB$ cannot violate any $r \in R$

and cannot entail any $n \in N$ either by monotonicity of $\mathcal{L}$. But it is possible that $\mathcal{K}_r^* \models QB$. So, validity of criterion 2) of this lemma is sufficient to guarantee that each diagnosis in $\mathbf{D}^0(Q)$ is in $\mathbf{D}^0(QB)$ as well, i.e. (II): $\mathbf{D}^0(QB) \supseteq \mathbf{D}^0(Q)$ holds.

As all diagnoses in $\mathbf{D}^+(Q)$ entail all formulas in Q by Definition 7.2, all diagnoses in $\mathbf{D}^+(Q)$ must entail QB as well. Consequently, due to deletion of some formulas from Q, no $\mathcal{D}_r \in \mathbf{D}^+(Q)$ can "move" to any set $\mathbf{D}^-(QB)$ or $\mathbf{D}^0(QB)$. That is, (III): $\mathbf{D}^+(QB) \supseteq \mathbf{D}^+(Q)$ must hold.

So, the overall conclusion is that, if criterion 1) and 2) are met, then (I), (II) and (III) hold. Assume that some $\supseteq$-relation in $i \in \{(I), (II), (III)\}$ is a $\supset$-relation. This leads to a violation of some $j \in \{(I), (II), (III)\}$ with $j \neq i$ since $\langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q) \rangle$ and $\langle \mathbf{D}^+(QB), \mathbf{D}^-(QB), \mathbf{D}^0(QB) \rangle$ are partitions of $\mathbf{D}$. Therefore, all $\supseteq$-relations must be $=$-relations and we can derive that $\mathfrak{P}(Q) = \mathfrak{P}(QB)$.

Moreover, we have that QB must be a query. This is due to the facts that QB is non-empty, Q is a query and the q-partitions of Q and QB are equal. Therefore, $\mathbf{D}^+(QB) = \mathbf{D}^+(Q) \geq 1$ and $\mathbf{D}^-(QB) = \mathbf{D}^-(Q) \geq 1$ which lets us conclude by Proposition 7.4 that QB is a query. $\square$

MINQ – The Divide-and-Conquer Strategy.   Intuitively, MINQ partitions the given query Q in two parts Q_1 and Q_2 and first analyzes Q_2 while Q_1 is part of QB (line 34). Note that in each iteration QB is the subset of Q that is currently assumed to be part of the sought minimized query (i.e. the one query that will finally be output by MINQ). In other words, analysis of Q_2 while Q_1 is part of QB means that all irrelevant formulas in Q_2 should be located and removed from Q_2 resulting in $Q_2^{\min} \subseteq Q_2$. That is, $Q_2^{\min}$ must include only relevant formulas which means that $Q_2^{\min}$ along with QB is a query with an equal q-partition as Q, but the deletion of any further formula from $Q_2^{\min}$ changes the q-partition.

After the relevant subset $Q_2^{\min}$ of Q_2, i.e. the subset that is part of the minimized query, has been returned, Q_1 is removed from QB, $Q_2^{\min}$ is added to QB and Q_1 is analyzed for a relevant subset that is part of the minimized query (line 35). This relevant subset, $Q_1^{\min}$, together with $Q_2^{\min}$, then builds a set-minimal subset of the input Q that is a query and has a q-partition equal to that of Q. Note that the argument X of MINQ is the subset of Q that has most recently been added to QB.

For each call in line 34 or line 35, the input Q to MINQ is recursively analyzed until a trivial case arises, i.e. (a) until Q is identified to be irrelevant for the computed minimized query wherefore $\emptyset$ is returned (lines 27 and 28) or (b) until $|Q| = 1$ and Q is not irrelevant for the computed minimized query wherefore Q is returned (lines 29 and 30).

Example 8.6   Let us reconsider the FOL DPI depicted by Table 15.2 on page 270. We recall that sets of minimal conflict sets and minimal diagnoses w.r.t. this DPI were given by $\mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} = \{\mathcal{C}_1, \mathcal{C}_2\} = \{\langle 1, 3, 4 \rangle, \langle 1, 2, 3, 5 \rangle\}$ as well as $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\} = \{[1], [3], [4, 5], [2, 4]\}$. For this DPI, a set of minimized queries computed by GETPOOLOFQUERIES is presented by Table 8.3. Note that these queries have been produced by different GETENTAILMENTS functions (as indicated by the dashed lines in Table 8.3). That is, Q_i for $i \in \{1, \dots, 5\}$ have been produced by the same GETENTAILMENTS function that is described in Example 8.1. For $i \in \{6, \dots, 9\}$, Q_i has been computed from a GETEN-TAILMENTS function that outputs only explicit entailments (cf. Definition 8.1) and Q_{10} from a GETEN-TAILMENTS function that returns a finite set of entailments where each entailment is some FOL formula. This could be accomplished, for example, by some resolution-based reasoning procedure [CL73].

It is important to realize that the results regarding Algorithm 4 established so far, most of which depend on the particular used GETENTAILMENTS function, must only hold within one part of Table 8.3 (where different parts are separated by the dashed lines). For example, for Q_2 and Q_9 it holds that $\mathbf{D}^+(Q_2) = \mathbf{D}^+(Q_9)$, but $\mathbf{D}^-(Q_2) \neq \mathbf{D}^-(Q_9)$ and $\mathbf{D}^0(Q_2) \neq \mathbf{D}^0(Q_9)$. By application of one and the same GETENTAILMENTS function, this case would be prohibited by Proposition 8.5. Furthermore, by Proposition 8.6, only Q_9 would be an element of the query pool $\mathbf{QP}$ in this case since $\mathbf{D}^0(Q_9) \subset \mathbf{D}^0(Q_2)$.

Moreover, we want to remark that Q_7, Q_8 and Q_9 can be seen as a proof that Q_6 is indeed set-

minimal. Each $Q_i, i \in \{7, 8, 9\}$ is a result of the removal of a single formula from Q_6. And, each such Q_i features a q-partition different from the one of Q_6. This illustrates quite well the principle of MINQ which performs tests of exactly this kind to verify minimality of a query or detect formulas that might be deleted from it under preservation of the q-partition, respectively.

Another essential note is that it is guaranteed that $\mathbf{D}^0(Q_6) = \emptyset$. This holds due to the construction of Q_6 as $U_{\mathbf{D}} \setminus \mathcal{D}_4 = \{1, 2, 3, 4, 5\} \setminus [2, 4] = \{1, 3, 5\}$ (recall that we use squared brackets to denote diagnoses in spite of the fact that these are sets, cf. Table 2.1). So, Q_6 comprises all formulas occurring in minimal diagnoses except for the ones contained in $\mathcal{D}_4$. We have that for any two different minimal diagnoses $\mathcal{D}_i, \mathcal{D}_j$ w.r.t. one and the same DPI it must be true that $\mathcal{D}_i \setminus \mathcal{D}_j \neq \emptyset$ as well as $\mathcal{D}_j \setminus \mathcal{D}_i \neq \emptyset$ as otherwise one would be necessarily a subset of the other. From this, we can easily derive that $\mathcal{K}_i^* \cup Q_6$ for $i \in \{1, \dots, 3\}$, i.e. for all minimal diagnoses $\mathcal{D}_i$ w.r.t. this DPI other than $\mathcal{D}_4$ which was used to build the query Q_6, must comprise a conflict set. This must be valid by the minimality of $\mathcal{D}_i$ and since by Q_6 at least one formula of $\mathcal{D}_i$ is readded to the KB. Note that a similar argumentation was used in the proof of Proposition 8.3. $\qquad \square$

i	Query Q_i	$\mathbf{D}^+(Q_i)$	$\mathbf{D}^-(Q_i)$	$\mathbf{D}^0(Q_i)$
1	$\{\forall X b(X) \to m_3(X)\}$	$\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_4\}$	$\{\mathcal{D}_3\}$	$\emptyset$
2	$\{b(w)\}$	$\{\mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_2\}$	$\{\mathcal{D}_1\}$
3	$\{\forall X m_1(X) \to b(X)\}$	$\{\mathcal{D}_1, \mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_2\}$	$\emptyset$
4	$\{m_1(w), m_2(u)\}$	$\{\mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_1\}$	$\emptyset$
5	$\{a(w)\}$	$\{\mathcal{D}_2\}$	$\{\mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_1\}$
6	$\{ax_1, ax_3, ax_5\}$	$\{\mathcal{D}_4\}$	$\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$	$\emptyset$
7	$\{ax_3, ax_5\}$	$\{\mathcal{D}_1, \mathcal{D}_4\}$	$\{\mathcal{D}_2, \mathcal{D}_3\}$	$\emptyset$
8	$\{ax_1, ax_5\}$	$\{\mathcal{D}_2, \mathcal{D}_4\}$	$\{\mathcal{D}_1, \mathcal{D}_3\}$	$\emptyset$
9	$\{ax_1, ax_3\}$	$\{\mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_1, \mathcal{D}_2\}$	$\emptyset$
10	$\{\forall X m_1(X) \to \neg a(X),$ $\forall X m_2(X) \to (\forall Y s(X, Y) \to a(Y))\}$	$\{\mathcal{D}_1\}$	$\{\mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}$	$\emptyset$

Table 8.3: Some queries and associated q-partitions for the DPI given by Table 15.2.

8.4 Soundness of Query Minimization

The following lemma shows that the function ISQPARTCONST used by MINQ is indeed a monotonic function (cf. Definition 4.6), which is a necessary prerequisite for versions of the QX algorithm to work in a sound way.

Lemma 8.2. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI, $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, $Q \in \mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ with q-partition $\mathfrak{P}(Q)$. Further, let $f : 2^Q \to \{0, 1\}$ be a function that maps a subset QB of Q to 1 if QB has q-partition $\mathfrak{P}(QB) = \mathfrak{P}(Q)$, to 0 otherwise. Then, f is a monotonic function (as per Definition 4.6).*

Proof. Assume a subset Q' of Q with $f(Q') = 1$, i.e. Q' has q-partition $\mathfrak{P}(Q') = \mathfrak{P}(Q)$. Let $Q' \subset Q'' \subseteq Q$ and assume that $f(Q'') = 0$, i.e. Q'' has a q-partition $\mathfrak{P}(Q'') \neq \mathfrak{P}(Q)$.

As shown in the proof of Lemma 8.1, $\mathbf{D}^+(X_1) \supseteq \mathbf{D}^+(X_2)$ holds for any $X_1 \subseteq X_2$. Therefore, we have $\mathbf{D}^+(Q') \supseteq \mathbf{D}^+(Q'') \supseteq \mathbf{D}^+(Q)$ and by $\mathfrak{P}(Q') = \mathfrak{P}(Q)$ that $\mathbf{D}^+(Q') = \mathbf{D}^+(Q)$ and thus that all $\supseteq$-relations are =-relations. So, either $\mathbf{D}^-(Q'') \neq \mathbf{D}^-(Q)$ or $\mathbf{D}^0(Q'') \neq \mathbf{D}^0(Q)$ must hold.

First, assume that $\mathbf{D}^-(Q'') \neq \mathbf{D}^-(Q)$. Then, as $\mathcal{K}_r^* \cup Q'' \subset \mathcal{K}_r^* \cup Q$ and by monotonicity of $\mathcal{L}$, it can only be the case that for some $\mathcal{D}_r \in \mathbf{D}$ some $x \in R \cup N$ that is violated for $\mathcal{K}_r^* \cup Q$ is *not*

violated for $\mathcal{K}_r^* \cup Q''$. Hence, $\mathbf{D}^-(Q'') \subset \mathbf{D}^-(Q)$ must hold. By a similar argumentation – *without* the assumption that $\mathbf{D}^-(Q') \neq \mathbf{D}^-(Q'')$ holds – we have that $\mathbf{D}^-(Q') \subseteq \mathbf{D}^-(Q'')$ and thus, altogether, that $\mathbf{D}^-(Q') \subset \mathbf{D}^-(Q)$ must be true. Due to $\mathfrak{P}(Q') = \mathfrak{P}(Q)$ we know that $\mathbf{D}^-(Q') = \mathbf{D}^-(Q)$ which is a contradiction.

Finally, assume that $\mathbf{D}^0(Q'') \neq \mathbf{D}^0(Q)$. Since $\mathcal{K}_r^* \cup Q$ does not violate any $x \in R \cup N$ for $\mathcal{D}_r \in \mathbf{D}^0(Q)$, $\mathcal{K}_r^* \cup Q''$ cannot violate any $x \in R \cup N$ by monotonicity of $\mathcal{L}$. As a conclusion, the only possibility for $\mathbf{D}^0(Q'') \neq \mathbf{D}^0(Q)$ is that $\mathcal{K}_r^* \models Q''$ for some $\mathcal{D}_r \in \mathbf{D}^0(Q)$, i.e. that $\mathcal{D}_r \in \mathbf{D}^+(Q'')$ which implies that $\mathbf{D}^0(Q'') \subset \mathbf{D}^0(Q)$. By a similar argumentation – *without* the assumption that $\mathbf{D}^0(Q') \neq \mathbf{D}^0(Q'')$ holds – we have that $\mathbf{D}^0(Q') \subseteq \mathbf{D}^0(Q'')$ and thus, altogether, that $\mathbf{D}^0(Q') \subset \mathbf{D}^0(Q)$ must be true. Due to $\mathfrak{P}(Q') = \mathfrak{P}(Q)$ we know that $\mathbf{D}^0(Q') = \mathbf{D}^0(Q)$ which is a contradiction.

This completes the proof for monotonicity of the given function f. $\qquad\square$

Proposition 8.7 (Correctness of MINQ). *Given a query $Q \in \mathbf{Q}_{\mathbf{D},\langle\mathcal{K},\mathcal{B},P,N\rangle_R}$ as input, MINQ computes a subset $Q_{\min} \subseteq Q$ such that $\mathfrak{P}(Q_{\min}) = \mathfrak{P}(Q)$ and there is no $Q' \subset Q_{\min}$ such that $\mathfrak{P}(Q') = \mathfrak{P}(Q)$.*

Proof. This proposition is a consequence of the correctness of QX shown by Proposition 4.9, of the correctness of function ISQPARTCONST established by Lemma 8.1 and of the monotonicity of the property tested by the function ISQPARTCONST guaranteed by Lemma 8.2. $\qquad\square$

8.5   Complexity of Query Pool Generation

The complexity of query minimization, i.e. one call to MINQ, in terms of calls to the ISQPARTCONST function is directly obtained from the complexity results for the standard QX algorithm given by Proposition 4.8.

Proposition 8.8 (Complexity of MINQ). *Let $\langle\mathcal{K},\mathcal{B},P,N\rangle_R$ be a DPI, $\mathbf{D} \subseteq \mathbf{mD}_{\langle\mathcal{K},\mathcal{B},P,N\rangle_R}$, $Q \in \mathbf{Q}_{\mathbf{D},\langle\mathcal{K},\mathcal{B},P,N\rangle_R}$ with $\mathfrak{P}(Q) = \langle\mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q)\rangle$ and the function SPLIT (line 31 of Algorithm 4) be defined as $\mathrm{SPLIT}(n) = \lfloor\frac{n}{2}\rfloor$ where n is a natural number. Then, the worst case number of calls to ISQPARTCONST during one call to MINQ$(\emptyset, Q, \emptyset, \mathfrak{P}(Q), \langle\mathcal{K},\mathcal{B},P,N\rangle_R)$ is in*

$$O\left(|Q_{\min}| \log \frac{|Q|}{|Q_{\min}|}\right)$$

where $Q_{\min}$ is the output of MINQ$(\emptyset, Q, \emptyset, \mathfrak{P}(Q), \langle\mathcal{K},\mathcal{B},P,N\rangle_R)$.

For any other definition of the function SPLIT, the worst case number of calls to ISQPARTCONST gets larger.

The overall complexity of GETPOOLOFQUERIES in terms of calls to functions that call the reasoner, i.e. functions GETENTAILMENTS, ISKBVALID and ISQPARTCONST, is established by the following proposition.

Proposition 8.9 (Complexity of GETPOOLOFQUERIES). *Let $\langle\mathcal{K},\mathcal{B},P,N\rangle_R$ be a DPI, q a natural number and $\mathbf{D} \subseteq \mathbf{mD}_{\langle\mathcal{K},\mathcal{B},P,N\rangle_R}$. Then, the worst case number of calls to functions that call a reasoner during one call to GETPOOLOFQUERIES$(\langle\mathcal{K},\mathcal{B},P,N\rangle_R, \mathbf{D}, q)$ is in*

$$O\left(\left(|\mathbf{D}| + \left|Q_{\min}^{(\max)}\right| \log \frac{\left|Q^{(\max)}\right|}{\left|Q_{\min}^{(\max)}\right|}\right) 2^{|\mathbf{D}|}\right)$$

where $\left|Q^{(\max)}\right|$ is the maximum size of a query before minimization, i.e. the size of the set of maximum cardinality that is stored in variable Q in line 19 throughout all iterations, and $\left|Q_{\min}^{(\max)}\right|$ is the maximum

size of a minimized query, i.e. the size of the set of maximum cardinality that is stored in variable Q' in line 19 throughout all iterations.

Proof. During the execution of the for-loop over lines 3-5 the function GETENTAILMENTS is called $|\mathbf{D}|$ times. During the execution of the for-loop over lines 6-22 which may be executed at most $2^{|\mathbf{D}|} - 2$ times, ISKBVALID is called at most $|\mathbf{D}| - 1$ times since $|\mathbf{S}| \geq 1$ and $\mathbf{S} \subset \mathbf{D}$ and thus $|\mathbf{D} \setminus \mathbf{S}| \leq |\mathbf{D}| - 1$ holds; furthermore, MINQ may be called once, namely if the condition tested by the if-statement in line 18 is true. During one execution of MINQ, by Proposition 8.8, at most

$$|Q_{\min}| \log \frac{|Q|}{|Q_{\min}|}$$

calls to ISQPARTCONST are made where $Q_{\min}$ is the output of the call to MINQ. So, an upper bound of the number of calls to ISQPARTCONST performed by one call to MINQ among all calls to MINQ throughout the execution of GETPOOLOFQUERIES, is

$$\left| Q_{\min}^{(\max)} \right| \log \frac{\left| Q^{(\max)} \right|}{\left| Q_{\min}^{(\max)} \right|}$$

where $\left| Q_{\min}^{(\max)} \right|$ is the set of maximum cardinality that is stored in variable Q' in line 19 throughout all iterations and $\left| Q^{(\max)} \right|$ is the set of maximum cardinality that is stored in variable Q in line 19 throughout all iterations.

So, all in all we know that functions that call a reasoner are invoked at most

$$|\mathbf{D}| + \left(|\mathbf{D}| - 1 + \left| Q_{\min}^{(\max)} \right| \log \frac{\left| Q^{(\max)} \right|}{\left| Q_{\min}^{(\max)} \right|} \right) (2^{|\mathbf{D}|} - 2)$$

times during the execution of GETPOOLOFQUERIES. Since

$$\left(|\mathbf{D}| + \left| Q_{\min}^{(\max)} \right| \log \frac{\left| Q^{(\max)} \right|}{\left| Q_{\min}^{(\max)} \right|} \right) 2^{|\mathbf{D}|}$$

is an upper bound of this number, the proposition holds. □

Note that none of the parameters that affect the complexity of the function GETPOOLOFQUERIES grows with the size of the DPI provided as an input to the interactive KB debugging problem. Merely the costs for reasoning, where a black-box debugging approach has no influence on, are affected by a higher complexity or larger size of the input DPI. Moreover, the size of the most relevant parameter influencing the worst case complexity, namely the exponent $|\mathbf{D}|$, can be specified by the user to any value greater or equal to 2. In other words, minus reasoning time, the generation of a pool of queries is a fixed parameter tractable problem [DF95] in the context of interactive KB debugging.

8.6 Shortcomings of Query Pool Generation

First, the exponential time complexity regarding the parameter $|\mathbf{D}|$ is a problem arising from the paradigm of computing an optimal query w.r.t. a certain quantitative measure $qsm()$ such as information gain [SFFR12, RSFF13] by calculating a (generally exponentially large) pool $\mathbf{QP}$ of queries in a first stage, whereupon

$qsm(Q) \in \mathbb{R}$ is evaluated for $Q \in \mathbf{QP}$ until the one Q^* with optimal $qsm(Q^*)$ is found and selected as the query to be asked to the user.

A key to solving this issue is the use of a different paradigm that does not rely on the computation of the pool $\mathbf{QP}$. Instead, qualitative measures can be derived from quantitative measures that have been used in interactive debugging scenarios [SFFR12, RSFF13, SF10]. These qualitative measures provide a way to estimate the $qsm()$ value of *partial q-partitions*, i.e. ones where not all leading diagnoses have been assigned to the respective set in the q-partition yet. That way a *direct* search for a query with (nearly) optimal properties is possible. A similar strategy called CKK has been employed in [SFFR12] for the information gain measure (see Section 9.3). From such a technique we can expect to save a high number of reasoner calls. Because only a usually small subset of q-partitions included in the pool computed by GETPOOLOFQUERIES is required to find a query with desirable properties if the search is implemented by means of a heuristic that involves the exploration of seemingly favorable (potential) queries and (partial) q-partitions, respectively, first. This is a topic of future work.

Another shortcoming of GETPOOLOFQUERIES is the extensive use of reasoning services which may be computationally expensive (depending on the given DPI). Instead of computing a set of common entailments Q of a set of KBs $\mathcal{K}_i^*$ first and consulting a reasoner to fill up the (q-)partition for Q in order to test whether Q is a query at all, the idea enabling a significant reduction of reasoner dependence is to compute some kind of *canonical query* without a reasoner and use simple set comparisons to decide whether the associated partition is a q-partition. Guided by qualitative properties mentioned before, a search for such q-partition with desirable properties can be accomplished *without reasoning at all*. Also, a set-minimal version of the optimal canonical query can be computed without reasoning aid. Only for the optional enrichment of the identified optimal canonical query by additional entailments and for the subsequent minimization of the enriched query, the reasoner may be employed. This is also a topic of future work.

Another aspect that can be improved is that *only one* minimized version of each query is computed by Algorithm 4. That is, per q-partition $\mathfrak{P}$, there might be some set-minimal queries which do not occur in the output set $\mathbf{QP}$. From the point of view of how well a query might be understood by an interacting user, of course not all minimized queries can be assumed equally good in general. Hence, in order to avoid a situation where a potentially best-understood query w.r.t. $\mathfrak{P}$ is not included in $\mathbf{QP}$, the query minimization process (see Section 8.3) might be adapted to take into account some information about faults the interacting user is prone to. This could be exploited to estimate how well this user might be able to understand and answer a query. For instance, given that the user frequently has problems to apply $\exists$ in a correct manner to express what they intend to express, but has never made any mistakes in formulating implications $\rightarrow$, then the query $Q_1 = \{\forall X\, p(X) \rightarrow q(X), r(a)\}$ might be better comprehended than $Q_2 = \{\forall X \exists Y s(X, Y)\}$. One way to achieve the finding of a well-understood query for some q-partition $\mathfrak{P}$ is to run the query minimization MINQ more than once, each time with a modified input (using a hitting set tree to accomplish this in a systematic manner – cf. Chapter 4, where an analogue idea is used to compute different minimal conflict sets w.r.t. a DPI). In this way, different set-minimal queries for $\mathfrak{P}$ can be identified and the process can be stopped when a suitable query is found.

8.7   Correctness of Query Pool Generation

The following proposition confirms the correctness of Algorithm 4, i.e. of the function GETPOOLOF-QUERIES. Roughly, it states that the output of $\mathbf{QP}$ of the function is duplicate-free, i.e. no query or q-partition occurs twice in $\mathbf{QP}$, that $\mathbf{QP}$ includes *only queries* and q-partitions, that tuples in $\mathbf{QP}$ are unique w.r.t. the set $\mathbf{D}^+$ of a q-partition and that, given $q > |\mathbf{QP}|$, there is no subset Y of $\mathbf{D}$ for which a q-partition with $\mathbf{D}^+ = Y$ exists and for which no q-partition with $\mathbf{D}^+ = Y$ is an element of $\mathbf{QP}$.

Proposition 8.10. *Let a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ such that $|\mathbf{D}| \geq 2$ and some $q \in \mathbb{N} \cup \{\infty\}$, $q \geq 1$ be the inputs to* GETPOOLOFQUERIES *and let $|\mathbf{QP}_{\max}| \geq 0$ be the maximum number of tuples $\langle Q, \mathfrak{P}(Q) \rangle$ that can be computed by* GETPOOLOFQUERIES *by means of the used* GETENTAIL-MENTS *function. If $q \geq |\mathbf{QP}_{\max}|$ (in particular $q = \infty$), then*

1. *there are no two tuples $\langle Q, \mathfrak{P}(Q) \rangle$, $\langle Q', \mathfrak{P}(Q') \rangle$ in $\mathbf{QP}$ such that $Q = Q'$ or $\mathfrak{P}(Q) = \mathfrak{P}(Q')$, and*

2. *$\mathbf{QP}$ includes a tuple $\langle Q, \langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q) \rangle \rangle$ only if $Q \in \mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, and*

3. *$\mathbf{QP}$ includes at most one tuple where $\mathbf{D}^+(Q) = Y$ for each $Y \subset \mathbf{D}$, and*

4. *for each $Y \subset \mathbf{D}$ for which a query Q w.r.t. $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ exists such that*

   (a) *Q includes only entailments computed by the used* GETENTAILMENTS *function and*

   (b) *$\mathfrak{P}(Q)$ is such that $\mathbf{D}^+(Q) = Y$,*

   $\mathbf{QP}$ includes a tuple $\langle Q', \mathfrak{P}(Q') \rangle$ such that $\mathbf{D}^+(Q') = Y$, and

5. *$\mathbf{QP} \neq \emptyset$.*

If $q < |\mathbf{QP}_{\max}|$, then $\mathbf{QP}$ includes q tuples satisfying (1), (2) and (3).

Proof. Statement (1) is a consequence of Proposition 8.2. Statement (2) is an implication of Proposition 8.1 and Proposition 8.7. The former says that only sets Q that are actually queries w.r.t. $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ can pass line 18. Thus, only queries are passed to MINQ as parameter Q. By the latter which states that MINQ is correct, i.e. outputs a query if the input is a query, statement (2) follows. Statement (3) follows from Proposition 8.5. If $q \geq |\mathbf{QP}_{\max}|$, the truth of statement (4) is witnessed by Proposition 8.4. Statement (5) is true by lines 23 and 24 and by Proposition 7.5 as well as Corollary 7.4 and the premise that $|\mathbf{D}| \geq 2$ which guarantee that the function ADDTRIVIALQUERIES always adds at least $|\mathbf{D}| \geq 2 > 0$ queries to $\mathbf{QP}$. In case $q < |\mathbf{QP}_{\max}|$, only statements (1), (2) and (3) are satisfied in general (for the same reasons as given above for the case $q \geq |\mathbf{QP}_{\max}|$) and $\mathbf{QP}$ is returned in line 22 by the definition of $|\mathbf{QP}_{\max}|$. Thence, the condition $|\mathbf{QP}| = q \geq 1$ tested in line 21 must be valid for $\mathbf{QP}$. $\square$

Algorithm 5 Interactive KB Debugging

Input: a tuple $\langle\langle\mathcal{K},\mathcal{B},P,N\rangle_R, n_{\min}, n_{\max}, t, p_{\widetilde{\mathcal{K}}\cup\overline{\mathcal{K}}}, q, qsm(), \sigma, mode\rangle$ consisting of

- an admissible DPI $\langle\mathcal{K},\mathcal{B},P,N\rangle_R$,
- leading diagnoses computation parameters, natural numbers $n_{\min}\geq 2, n_{\max}, t$,
- a function $p_{\widetilde{\mathcal{K}}\cup\overline{\mathcal{K}}}:\widetilde{\mathcal{K}}\cup\overline{\mathcal{K}}\to(0,1]$,
- a parameter $q\in\mathbb{N}\cup\{\infty\}, q\geq 1$ that determines the size of the computed query pool,
- a function $qsm(Q)\in\mathbb{R}$ used for query selection that assigns a real number to a query Q to express the "goodness" of Q,
- a maximum fault tolerance $\sigma\in[0,1]$ and
- a mode $mode\in\{static, dynamic\}$ that determines the used method for diagnosis computation.

Output: The output depends on $mode$ and σ:

- $mode=static$: a maximal solution KB w.r.t. the input DPI $\langle\mathcal{K},\mathcal{B},P,N\rangle_R$ which is
  - an approximation of the solution to Interactive Static KB Debugging (Problem Def. 6.2) if $\sigma>0$.
  - the (exact) solution to Interactive Static KB Debugging if $\sigma=0$.
- $mode=dynamic$: a maximal solution KB w.r.t. the current DPI $\langle\mathcal{K},\mathcal{B},P\cup P',N\cup N'\rangle_R$ which is
  - an approximation of the solution to Interactive Dynamic KB Debugging (Problem Def. 6.1) if $\sigma>0$.
  - the (exact) solution to Interactive Dynamic KB Debugging if $\sigma=0$.

(for a more formal and precise characterization of the output see Proposition 9.1 on page 124)

1: $P', N', \mathbf{C}_{calc}, \mathbf{D}_{\checkmark}, \mathbf{D}_{\times}, \mathbf{D}_{out}, \mathbf{D}_{\supset}, qData\leftarrow\emptyset$
2: $\mathbf{Q}_{dup}, QA\leftarrow[\,]$
3: $\mathbf{Q}\leftarrow[\emptyset]$
4: $answer\leftarrow false$
5: $p_{\mathcal{K}}()\leftarrow\text{GETFORMULAPROBS}(\mathcal{K}, p_{\widetilde{\mathcal{K}}\cup\overline{\mathcal{K}}}())$ ▷ application of Formulas 4.2 and 4.7
6: **while** $true$ **do**
7:     **if** $mode=static$ **then**   ▷ see Algorithm 7
8:         $\langle\mathbf{D}_{\checkmark},\mathbf{Q},\mathbf{C}_{calc},\mathbf{D}_{\times}\rangle\leftarrow\text{STATICHS}(\langle\mathcal{K},\mathcal{B},P,N\rangle_R,\mathbf{Q},t,n_{\min},n_{\max},$
                                      $\mathbf{C}_{calc},\mathbf{D}_{\checkmark},\mathbf{D}_{\times},p_{\mathcal{K}}(),P',N')$
9:     **else**   ▷ see Algorithms 8, 9 and 10
10:         $\langle\mathbf{D}_{\checkmark},\mathbf{Q},\mathbf{C}_{calc},\mathbf{D}_{\times},\mathbf{D}_{\supset},\mathbf{Q}_{dup}\rangle\leftarrow\text{DYNAMICHS}(\langle\mathcal{K},\mathcal{B},P,N\rangle_R,\mathbf{Q},\mathbf{Q}_{dup},t,n_{\min},n_{\max},$
                                      $\mathbf{C}_{calc},\mathbf{D}_{\checkmark},\mathbf{D}_{\times},p_{\mathcal{K}}(),P',N',\mathbf{D}_{\supset})$
11:     $p_{\mathbf{D}}()\leftarrow\text{GETPROBDIST}(\mathbf{D}_{\checkmark},p_{\mathcal{K}}(),\langle\mathcal{K},\mathcal{B},P,N\rangle_R,QA)$ ▷ see Algorithm 6
12:     $\mathcal{D}_{\max}\leftarrow\text{GETMODE}(\mathbf{D}_{\checkmark},p_{\mathbf{D}}())$
13:     **if** $p_{\mathbf{D}}(\mathcal{D}_{\max})\geq 1-\sigma$ **then**   ▷ stop criterion
14:         **return** $\text{GETSOLKB}(\mathcal{D}_{\max},\langle\mathcal{K},\mathcal{B},P\cup P',N\cup N'\rangle_R,P',mode)$ ▷ return solution KB
15:     **else**
16:         $\langle Q,\mathfrak{P}(Q)\rangle\leftarrow\text{CALCQUERY}(\mathbf{D}_{\checkmark},qData,p_{\mathbf{D}}(),p_{\widetilde{\mathcal{K}}\cup\overline{\mathcal{K}}}(),qsm(),$
                                      $\langle\mathcal{K},\mathcal{B},P\cup P',N\cup N'\rangle_R,q)$ ▷ see Algorithm 6
17:         $answer\leftarrow u(Q)$ ▷ user interaction
18:         $QA\leftarrow\text{APPEND}(\langle Q,answer\rangle,QA)$
19:         $\mathbf{D}_{out}\leftarrow\text{GETINVALIDDIAGS}(\mathfrak{P}(Q),answer)$
20:         $qData\leftarrow\text{UPDATEQDATA}(\mathbf{D}_{out},\mathbf{D}_{\checkmark},answer)$
21:         $\mathbf{D}_{\checkmark}\leftarrow\mathbf{D}_{\checkmark}\setminus\mathbf{D}_{out}$
22:         $\mathbf{D}_{\times}\leftarrow\mathbf{D}_{\times}\cup\mathbf{D}_{out}$
23:         **if** $answer=true$ **then**
24:             $P'\leftarrow P'\cup\{Q\}$
25:         **else**
26:             $N'\leftarrow N'\cup\{Q\}$

Algorithm 6 Interactive KB Debugging (continued)

27: **procedure** GETPROBDIST($\mathbf{D}_\checkmark, p_\mathcal{K}(), \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R, QA$)
28:     $P'', N'' \leftarrow \emptyset$
29:     $p_{\mathbf{D}, prio}() \leftarrow$ GETPRIODIAGPROBS($\mathbf{D}_\checkmark, p_\mathcal{K}(), \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$)   ▷ application of Formula 4.3
30:     **for** $\langle Q, u(Q) \rangle \in QA$ **do**   ▷ run through chronologically sorted query-answer pairs
31:         **if** $u(Q) = true$ **then**
32:             **for** $\mathcal{D}_r \in \mathbf{D}_\checkmark$ **do**   ▷ function GETENTAILMENTS is defined on page 104
33:                 $E_{\mathcal{D}_r} \leftarrow$ GETENTAILMENTS($\mathcal{D}_r, \mathcal{K}, \mathcal{B}, P \cup P''$)   ▷ $E_{\mathcal{D}_r}$ is a set of entailments of $\mathcal{K}_r^*$
34:                 **if** $Q \not\subseteq E_{\mathcal{D}_r}$ **then**   ▷ $\mathcal{D}_r \in \mathbf{D}^0(Q)$
35:                     $p_{\mathbf{D}, prio}(\mathcal{D}_r) \leftarrow \frac{1}{2} p_{\mathbf{D}, prio}(\mathcal{D}_r)$
36:             $P'' \leftarrow P'' \cup \{Q\}$
37:         **else**
38:             **for** $\mathcal{D}_r \in \mathbf{D}_\checkmark$ **do**   ▷ ISKBVALID (see Algorithm 1)
39:                 **if** ISKBVALID($(\mathcal{K} \setminus \mathcal{D}_r) \cup Q, \langle \cdot, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R$) **then**   ▷ $\mathcal{D}_r \in \mathbf{D}^0(Q)$
40:                     $p_{\mathbf{D}, prio}(\mathcal{D}_r) \leftarrow \frac{1}{2} p_{\mathbf{D}, prio}(\mathcal{D}_r)$
41:             $N'' \leftarrow N'' \cup \{Q\}$
42:     $sum \leftarrow \sum_{\mathcal{D}_r \in \mathbf{D}_\checkmark} p_{\mathbf{D}, prio}(\mathcal{D}_r)$
43:     **for** $\mathcal{D}_r \in \mathbf{D}_\checkmark$ **do**
44:         $p_{\mathbf{D}, prio}(\mathcal{D}_r) \leftarrow \frac{1}{sum} p_{\mathbf{D}, prio}(\mathcal{D}_r)$ ▷ normalization
45:     **return** $p_{\mathbf{D}, prio}()$

46: **procedure** CALCQUERY($\mathbf{D}_\checkmark, qData, p_\mathbf{D}(), p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}(), qsm(), \langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R, q$)
47:     $\mathbf{QP} \leftarrow$ GETPOOLOFQUERIES($\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R, \mathbf{D}_\checkmark, q$)   ▷ see Algorithm 4
48:     **return** SELECTBESTQUERY($\mathbf{QP}, qData, p_\mathbf{D}(), p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}(), qsm()$)   ▷ see Section 9.3

Chapter 9

An Algorithm for Interactive Knowledge Base Debugging

In this chapter we will give a description of an algorithm for interactive KB debugging (Algorithm 5) which implements the entire functionality required by an interactive debugging system. All other algorithms presented so far will be subroutines of Algorithm 5 which are either directly or indirectly called by it. Before we explain and discuss Algorithm 5 in detail, we give the reader a rough and informal overview of the algorithm's input, output and actions in the following section in order to make the details of the algorithm easier to digest.

Remark 9.1 Note, in the following, when we speak of *the input DPI* we refer to the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that is provided as an input to Algorithm 5, by *the current DPI* we mean the DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ where P' and N', respectively, are all positive and negative test cases added to the input DPI from the start of the algorithm's execution until the current point in time. Further on, *an intermediate (or previous) DPI* denotes a DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R$ which is not the current DPI and where $\emptyset \subseteq P'' \subseteq P'$ and $\emptyset \subseteq N'' \subseteq N'$. Finally, *the last-but-one DPI* corresponds to an intermediate DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R$ where either $|P'| = |P''| + 1$ or $|N'| = |N''| + 1$ is true, but not both. $\square$

9.1 Interactive Debugging Algorithm: Overview

Input:
An admissible DPI and some meta information where the latter consists of

- fault probabilities of syntactical elements occurring in the KB,

- a minimal and desired number of leading diagnoses,

- a desired maximum reaction time (time between two successive queries presented to the user),

- a maximum fault tolerance (roughly, the probability of being presented a non-desired solution KB as output),

- a measure for query selection (determines which query is the best query within a given set of queries),

- a parameter that determines the size of the computed pool of queries in each iteration and

- a parameter specifying the way the hitting set tree for computation of leading diagnoses is constructed and updated.

Output:

A solution KB such that the diagnosis used to formulate the solution KB has a probability (w.r.t. the current leading diagnoses) greater than or equal to 1 minus the given maximum fault tolerance.

Procedure:

1. *Initialization:* Compute the fault probability of each formula in the KB by means of the given fault probabilities.

2. *Leading Diagnoses Computation:* Use a hitting set tree constructed and updated in a manner as specified in the input coupled with QX to calculate a set of leading diagnoses. In that, the cardinality and computation time of the set of leading diagnoses is determined by the corresponding input parameters specifying minimal and desired number of leading diagnoses and desired reaction time.

3. *Probability Update and Stop Criterion:* Use the formula fault probabilities and the new information obtained by already specified test cases (answered queries) to compute updated (posterior) probabilities of the current leading diagnoses. If one diagnosis probability is greater than or equal to 1 minus the maximum fault tolerance, return the solution KB obtained by deletion of this diagnosis from the KB and subsequent addition of the union of all positive test cases.

4. *Query Generation and Selection:* Use the set of leading diagnoses (and possibly their fault probabilities) to generate a pool of queries, the size of which depends on the respective parameter provided as input. Given the pool of queries, select the best query according to the given query selection measure.

5. *User Interaction and Incorporation of New Information:* Ask the user the selected query and add it to the positive test cases in case of a positive answer and to the negative test cases otherwise.

6. *Hitting Set Tree Update:* Update the hitting set tree based on the new information given by the classification of the test case resulting from the query answer. In particular, this involves the deletion of all those minimal diagnoses that conflict with the new test case.

7. Repeat from Step 2.

9.2 Interactive Debugging Algorithm: Detailed Description

To describe the detailed process of Algorithm 5, we first characterize the input arguments, the output and the meaning of the variables used and then provide a step-by-step textual description of the actions taken by the algorithm.

9.2.1 Input Arguments

The input parameters of Algorithm 5 are the following:

- An admissible DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ (cf. Definition 3.6).

- Natural numbers $n_{\min} \geq 2, n_{\max}, t$ for leading diagnoses calculation (see description in Chapter 7 on page 95).

  Remark: The postulation $n_{\min} \geq 2$ is necessary in order for the existence of queries w.r.t. any computed set of leading minimal diagnoses $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ to be guaranteed (see Proposition 7.5).

- A function $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}} : \widetilde{\mathcal{K}} \cup \overline{\mathcal{K}} \to (0,1]$ that assigns a fault probability $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}(e)$ to each $e \in \widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}$ reflecting the degree of belief that (one occurrence of) a syntactical element e appearing in $\mathcal{K}$ is faulty (see Section 4.6).

  Remarks: Forbidding a probability of zero for syntactical elements assures that no formula in $\mathcal{K}$ can have a probability of zero (cf. Remark 4.5).

  Recall from Section 4.6.1 that $\widetilde{\mathcal{K}}$ refers to the signature of $\mathcal{K}$ (cf. Chapter 2) and $\overline{\mathcal{K}}$ denotes the set of all logical connectives occurring in $\mathcal{K}$. From probabilities of logical connectives and elements of the signature, probabilities of formulas in $\mathcal{K}$ and from those in turn probabilities of diagnoses w.r.t. the DPI can be derived as shown by Formulas 4.2 and 4.3.

  Further note that in the description of the algorithms in this section, unlike in Section 4.6, we use different denotations for probabilities of syntactical elements ($p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}$), formulas ($p_{\mathcal{K}}()$) and diagnoses ($p_{\mathbf{D}}()$) in order to make a clear distinction between these different functions.

- A natural number $q \geq 1$ that denotes the number of queries that should be precomputed, i.e. the preferred size of the query pool $\mathbf{QP}$ (see Chapter 8), before the "best" tuple $\langle Q^*, \mathfrak{P}(Q^*) \rangle$ is selected from $\mathbf{QP}$.

  Remark: In general, higher q implies better quality of the selected query in terms of the query selection measure $qsm()$ (see next bullet point). The chance of locating a good query in a larger set of queries is higher. On the other hand, higher q involves a worse reaction time, i.e. time between two successive queries. The more queries are computed, the more time the function GETPOOLOF-QUERIES consumes.

- A query selection measure $qsm()$ where $qsm : \mathbf{QP} \to \mathbb{R}$ is a function that assigns a real-valued number $qsm(\langle Q, \mathfrak{P}(Q) \rangle)$ to each tuple in $\mathbf{QP}$, often called the score of $\langle Q, \mathfrak{P}(Q) \rangle$.

  Remark: $qsm()$ defines what is considered the "best" query in the set $\mathbf{QP}$, namely the query Q^* in the tuple $\langle Q^*, \mathfrak{P}(Q^*) \rangle$ with best score among all tuples in the pool $\mathbf{QP}$. Diverse measures that can be used as a $qsm()$ function in this algorithm have been discussed and evaluated within the scope of interactive KB debugging in literature [SFFR12, RSFF13] (for details see Section 9.3).

- A maximum fault tolerance σ that defines the stop criterion of the algorithm. That is, for a current set of leading diagnoses, the stop criterion is satisfied iff the most probable leading diagnosis has an (updated) probability of at least $1 - \sigma$ (see below for a precise definition of what "updated" means).

  Remark: The smaller σ is chosen, the higher is the chance that a desired diagnosis is found. Selecting $\sigma := 0$, i.e. admitting zero fault tolerance, is the safest (but also most time-consuming) way to run a debugging session with Algorithm 5, as in this case the session will stop only after all but one diagnosis have been invalidated by test cases.

- A mode $mode \in \{static, dynamic\}$ that determines

  (i) which type of leading diagnoses are computed, i.e. only minimal diagnoses w.r.t. the input DPI ($static$) or minimal diagnoses w.r.t. the current DPI ($dynamic$),

  (ii) the hitting set tree pruning strategy after a query has been answered, i.e. conservative pruning ($static$) or invasive pruning ($dynamic$),

  (iii) the space and time complexity of diagnosis computation, i.e. not much affected by the asked queries ($static$) – tree is almost monotonically growing, but cannot get larger in size than the complete non-interactive hitting set tree (the tree produced by Algorithm 2 with input $n_{\min} = \infty$) – or significantly influenced by the asked queries ($dynamic$) – tree may shrink significantly if new test cases do not introduce "completely new" minimal conflict sets (that

are in no subset-relation with an existing one), or lead to a tree that is significantly larger than
the complete non-interactive hitting set tree if many "completely new" minimal conflict sets
result from the addition of new test cases. For an in-depth discussion and comparison of both
strategies the reader may consult Part III.

9.2.2   Output

The output of Algorithm 5 can be explained as follows by making a distinction between the two modes
of the algorithm specified by input parameter $mode$:

Proposition 9.1. *If $mode = static$, then Algorithm 5 returns the (exact) solution of the Interactive Static
KB Debugging problem (Problem Definition 6.2) if $\sigma = 0$ and an approximate solution of the problem if
$\sigma > 0$ where the likeliness of finding the (exact) solution increases with decreasing σ.*

*More concretely, a maximal solution KB $\mathcal{K}^* = (\mathcal{K} \setminus \mathcal{D}_{\max}) \cup U_P$ w.r.t. the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$
is returned such that*

1. $\mathcal{D}_{\max} \in \mathbf{D}$ *($\mathcal{D}_{\max}$ is an element of the current set of leading diagnoses)*

2. $\mathcal{D}_{\max} = \arg\max_{\mathcal{D} \in \mathbf{D}} p_{\mathbf{D}}(\mathcal{D})$ *($\mathcal{D}_{\max}$ is the a-posteriori most probable leading diagnosis)*

3. $p_{\mathbf{D}}(\mathcal{D}_{\max}) \geq 1 - \sigma$ *(the a-posteriori probability of $\mathbf{D}_{\max}$ exceeds the predefined threshold)*

4. $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ *comprises the $|\mathbf{D}|$ most probable minimal diagnoses
w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ as per the diagnosis probability measure $p_{\mathbf{D},prio}()$*
   *(the set of leading diagnoses corresponds to the a-priori most probable minimal diagnoses w.r.t.
   the input DPI that satisfy all specified test cases),*

5. *a-priori probability measure $p_{\mathbf{D},prio}()$ is computed from $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}()$ as per*

   (a) *Formula 4.2*   *(computation of formula fault probabilities)*

   (b) *Formula 4.7*   *(adaptation of formula fault probabilities)*

   (c) *Formula 4.3*   *(computation of diagnoses probabilities from formula fault probabilities)*

6. *the a-posteriori probability measure $p_{\mathbf{D}}()$ is computed from $p_{\mathbf{D},prio}()$ as per Bayes' Theorem (For-
   mula 4.5, for details see below) taking into account the new information given by the set of all
   answered queries so far, i.e. the collected sets of positive (P') and negative (N') test cases.*

*If $mode = dynamic$, then Algorithm 5 returns the (exact) solution of the Interactive Dynamic KB De-
bugging problem (Problem Definition 6.1) if $\sigma = 0$ and an approximate solution of the problem if $\sigma > 0$
where the likeliness of finding the (exact) solution increases with decreasing σ.*

*More concretely, a maximal solution KB $\mathcal{K}^* = (\mathcal{K} \setminus \mathcal{D}_{\max}) \cup U_{P \cup P'}$ w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup
P', N \cup N' \rangle_R$ is returned such that*

1. $\mathcal{D}_{\max} \in \mathbf{D}$ *($\mathcal{D}_{\max}$ is an element of the current set of leading diagnoses)*

2. $\mathcal{D}_{\max} = \arg\max_{\mathcal{D} \in \mathbf{D}} p_{\mathbf{D}}(\mathcal{D})$ *($\mathcal{D}_{\max}$ is the a-posteriori most probable leading diagnosis)*

3. $p_{\mathbf{D}}(\mathcal{D}_{\max}) \geq 1 - \sigma$ *(the a-posteriori probability of $\mathbf{D}_{\max}$ exceeds the predefined threshold)*

4. $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ *comprises the $|\mathbf{D}|$ most probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup
   P', N \cup N' \rangle_R$ as per the diagnosis probability measure $p_{\mathbf{D},prio}()$*
   *(the set of leading diagnoses corresponds to the a-priori most probable minimal diagnoses w.r.t.
   the current DPI),*

5. *the a-priori probability measure $p_{\mathbf{D},prio}()$ is computed from $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}()$ as per*

   (a) *Formula 4.2*   *(computation of formula fault probabilities)*

   (b) *Formula 4.7*   *(adaptation of formula fault probabilities)*

   (c) *Formula 4.3*   *(computation of diagnoses probabilities from formula fault probabilities)*

6. *the a-posteriori probability measure $p_{\mathbf{D}}()$ is computed from $p_{\mathbf{D},prio}()$ as per Bayes' Theorem (Formula 4.5, for details see below) taking into account the new information given by the set of all answered queries so far, i.e. the collected sets of positive (P') and negative (N') test cases.*

Remark 9.2 We still need to explain what we mean by "approximate solution" of the Interactive Static (Dynamic) KB Debugging problem. Roughly, an approximate solution is one constructed from a diagnosis which is not the only remaining minimal diagnosis. More precisely, an *approximate solution* of

- the Interactive Static KB Debugging problem is a maximal solution KB $(\mathcal{K} \setminus \mathcal{D}) \cup U_P$ such that

  – $\mathcal{D}$ is a minimal diagnosis w.r.t. the input DPI and w.r.t. the current DPI and

  – there is some $\mathcal{D}' \neq \mathcal{D}$ which is a minimal diagnosis w.r.t. the input DPI and w.r.t. the current DPI

- the Interactive Dynamic KB Debugging problem is a maximal solution KB $(\mathcal{K} \setminus \mathcal{D}) \cup U_{P \cup P'}$ such that

  – $\mathcal{D}$ is a minimal diagnosis w.r.t. the current DPI and

  – there is some $\mathcal{D}' \neq \mathcal{D}$ which is a minimal diagnosis w.r.t. the current DPI

where the input DPI is given by $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and the currect DPI by $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$.

So, as long as not all but one diagnosis candidate that enables the formulation of a solution KB has been ruled out by the classification of test cases, we speak of an approximate solution. Now, the lower a value for σ is predefined, the longer Algorithm 5 will usually need to iterate and the more test cases will usually need to be specified until one diagnosis has a probability greater than or equal to $1 - \sigma$. Thence, at the time a diagnosis exceeds the probability $1 - \sigma$ there will be usually fewer minimal diagnoses left than in case of the selection a higher value for σ. Therefore, the likeliness of picking the (exact) solution will usually be the higher, the lower σ is.   □

Remark 9.3 Note that granting a maximum *absolute* fault tolerance σ that is independent of a set of leading diagnoses is generally computationally infeasible due to the high complexity of diagnosis computation (see Chapter 1). Since, for an absolute fault tolerance to hold, *all* minimal diagnoses w.r.t. the current DPI have to be computed in order to determine their probability and to decide whether the most probable diagnosis has a probability greater than or equal to $1 - \sigma$.

In fact, the fault tolerance used by Algorithm 5 which is *relative* to the set of leading diagnoses, i.e. the (a-priori) most probable minimal diagnoses $\mathbf{D}$ w.r.t. a DPI can be interpreted as follows. Under the assumption that the true diagnosis $\mathcal{D}_t$ is included in $\mathbf{D}$, the chance that the most probable minimal diagnosis $\mathcal{D}_{\max} \in \mathbf{D}$ which satisfies the stop criterion is not equal to $\mathcal{D}_t$ is smaller than the predefined threshold σ (cf. Section 4.6). Thus, under this assumption, the (a-posteriori) probability of being presented a non-desired solution KB as output of Algorithm 5 is smaller than σ.

The a-priori diagnoses probability measure $p_{\mathbf{D},prio}()$ refers to the one that is computed *directly* from the fault information provided as an input to Algorithm 5 whereas the a-posteriori diagnoses probability measure $p_{\mathbf{D}}()$ is the one obtained from $p_{\mathbf{D},prio}()$ after incorporating the information given by the new

test cases specified so far during the debugging session. So, $p_{\mathbf{D},prio}()$ and $p_{\mathbf{D}}()$ might differ in terms of the probability order of diagnoses. Incorporation of updated probabilities *directly* into the hitting set tree algorithms to be used for the determination of leading diagnoses in the order prescribed by an updated probability measure is only possible if there is an additional update operator (besides Bayes' Theorem for adapting *diagnoses* probabilities) that can be applied to *formula* probabilities. For, the latter are exploited in the hitting set tree to assign probability weights to paths that are *not yet diagnoses* (cf. $p_{nodes}()$ specified by Definition 4.9 and the discussion of Formula 4.6) in order to guide the search for minimal diagnoses in best-first order. Updated *diagnosis* probabilities are not helpful at all for this purpose. Devising a reasonable mechanism of updating formula probabilities seems to be hard mostly due to the lack of suitable data that might be collected during the debugging session to accomplish that. What would be imaginable during the debugging session is to try to learn something about the fault probability of *syntactical elements* by examining the positive (all formulas are *definitely* correct) and singleton negative (the single formula is *definitely* incorrect) test cases. However, a drawback of such a strategy comes into effect when only syntactically very simple queries are used which is, for instance, the case in Example 8.1 (see the definition of the GETENTAILMENTS function there). From such queries not many useful insights concerning faulty syntactical elements might be gained. On the other hand, such queries are absolutely desirable from the point of view of how well a user might comprehend the formulas asked by the system. Hence, these two aspects seem to contradict each other. Still, it is a topic for future research to attempt to elaborate a solution for that issue.

A way to achieve that $p_{\mathbf{D}}()$ coincides with $p_{\mathbf{D},prio}()$, at least in case $mode = static$, is to exclude queries Q with $\mathbf{D}^0(Q) \neq \emptyset$ (see Remark 9.8). How this might be accomplished is stated by Proposition 8.3. Please notice that ignorance of queries with non-empty $\mathbf{D}^0$ does not implicate any disadvantages for interactive debugging. On the contrary, it is even a desirable feature of a debugger and brings along higher computational efficacy of query generation and stronger test cases from the logical point of view (cf. Section 8.2). For the scenario $mode = dynamic$, it is not possible in general to bypass the probability update by means of such queries (see Remark 9.8). □

9.2.3 Variables

The variables used by Algorithm 5 that are not input arguments to the algorithm are the following:

- P', N' are the sets of positive and negative test cases, respectively, collected *during* the execution of Algorithm 5 so far. That is, P' stores all positively answered queries, whereas N' stores all negatively answered ones.

- $\mathbf{C}_{calc}$ is the set of all conflict sets computed by QX during the execution of Algorithm 5 so far.

  Remark: In case of static debugging ($mode = static$), $\mathbf{C}_{calc}$ includes exclusively minimal conflict sets w.r.t. the input DPI, whereas, in case of dynamic debugging ($mode = dynamic$), $\mathbf{C}_{calc}$ may comprise minimal conflict sets w.r.t. the current or any intermediate DPI.

- $\mathbf{D}_{\checkmark}$ is the set of leading diagnoses returned by a call of STATICHS in case of static debugging ($mode = static$) and by a call of DYNAMICHS in case of dynamic debugging ($mode = dynamic$).

  Remarks: In case of dynamic debugging, $\mathbf{D}_{\checkmark} \subseteq \mathbf{mD}_{\langle \mathcal{K},\mathcal{B},P\cup P',N\cup N'\rangle_R}$ is the set of most probable minimal diagnoses w.r.t. the current DPI $\langle \mathcal{K},\mathcal{B}, P \cup P', N \cup N'\rangle_R$ as per the diagnosis probability measure $p_{\mathbf{D},prio}()$ computed from $p_{\widetilde{\mathcal{K}}\cup\overline{\mathcal{K}}}()$ by Formulas 4.2, 4.7, 4.3 and 4.4 (cf. Sections 4.6 and 9.2.2).

  In case of static debugging, $\mathbf{D}_{\checkmark} \subseteq \mathbf{mD}_{\langle \mathcal{K},\mathcal{B},P\rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K},\mathcal{B},P\cup P',N\cup N'\rangle_R}$, i.e. $\mathbf{D}_{\checkmark}$ includes only diagnoses that are minimal diagnoses w.r.t. the input DPI $\langle \mathcal{K},\mathcal{B}, P, N\rangle_R$ as well as w.r.t. the current DPI $\langle \mathcal{K},\mathcal{B}, P\cup P', N\cup N'\rangle_R$. Moreover, $\mathbf{D}_{\checkmark}$ comprises the most probable minimal diagnoses *w.r.t.*

the input DPI according to the diagnosis probability measure $p_{\mathbf{D},prio}()$ computed from $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}()$ by Formulas 4.2, 4.7, 4.3 and 4.4 (cf. Sections 4.6 and 9.2.2).

- $\mathbf{D}_\times$ stores all minimal diagnoses w.r.t. the input DPI that have been invalidated by one of the collected positive and negative test cases P' and N', respectively ($mode = static$). $\mathbf{D}_\times$ stores the minimal diagnoses w.r.t. the last-but-one DPI that have been invalidated by the most recently added test case ($mode = dynamic$).

- $\mathbf{D}_{out}$ is the subset of the set of current leading diagnoses $\mathcal{D}_\checkmark$ that has been invalidated by the most recently added test case.

- $\mathbf{D}_\supset$ stores all diagnoses that are non-minimal w.r.t. the current DPI, i.e. for each diagnosis $\mathsf{nd} \in \mathbf{D}_\supset$ there is some $\mathsf{nd}' \in \mathbf{D}_\checkmark$ such that $\mathsf{nd} \supset \mathsf{nd}'$ ($mode = dynamic$).

  Remark: $\mathbf{D}_\supset$ is solely needed for dynamic and not for static debugging as the latter does not need to store non-minimal diagnoses (cf. rule 4 of Definition 4.8 on page 59). Reason for this is the fact that only minimal diagnoses w.r.t. the input DPI are searched for. On the other hand, in case of dynamic debugging, non-minimal diagnoses might become minimal ones after some new test cases are specified since minimal diagnoses w.r.t. the (changing) current DPI are considered.

- $qData$ is an informal variable that comprehends any kind of data that might be taken into account by the query selection measure $qsm()$ and that might need to be adapted after a query has been answered (and diagnoses have been invalidated) in order to take the obtained new information into account. One can imagine $qData$ as a log specific to the particular function $qsm()$ that is used which records data of prior (query answering) iterations executed by the algorithm such as certain performance measures. An example of a $qsm()$ strategy using one such metric, namely the ratio of leading diagnoses invalidated by a test case, can be found in [RSFF13].

- $QA := [\langle Q, u(Q) \rangle]_{Q \in P' \cup N'}$ where $u(Q) \in \{true, false\}$ is the chronologically ordered list of queries and user answers collected so far during the execution of Algorithm 5.

- $\mathbf{Q}$ is the current queue of open nodes in the hitting set tree maintained by Algorithm 5.

- The list $\mathbf{Q}_{dup}$ roughly stores all duplicate nodes (that is, nodes for each of which there is a node in the hitting set tree that corresponds to an equal set of edge labels) computed so far during the execution of Algorithm 5.

  Remark: The list $\mathbf{Q}_{dup}$ is only relevant in case $mode = dynamic$ and not needed if $mode = static$. The purpose of this set is to enable the "replacement" of pruned nodes which is necessary to guarantee the completeness of DYNAMICHS in terms of not missing any minimal diagnoses (for a detailed explanation, see Chapter 12).

9.2.4 Algorithm Walkthrough

Initialization. In the first 4 lines, variable declarations take place. First, all variables that store sets of conflict sets, diagnoses or test cases, and $qData$ are initialized to the empty set. Further on, $\mathbf{Q}_{dup}$ and QA are initialized to an empty list. Finally, the queue $\mathbf{Q}$ of open nodes used for the hitting set tree construction by STATICHS ($mode = static$) or DYNAMICHS ($mode = dynamic$), respectively, is set to $[\emptyset]$ since it initially includes only a non-labeled root node.

Remark 9.4 The non-labeled root node is denoted by $\emptyset$ since nodes in STATICHS are associated with the set of edge labels along the path in the hitting set tree from the root node to this node (cf. Chapters 4 and 11). Hence, the root node itself corresponds to the empty path which includes no edges.

Notice that in case of DYNAMICHS, nodes will be (ordered) lists instead of (non-ordered) sets like in STATICHS (cf. Chapter 12). That is, to be precise, the unlabeled root node in this case corresponds to the empty list $[]$. For the ease of representation of Algorithm 5, only one set $\mathbf{Q}$ is initialized to be used with either STATICHS or DYNAMICHS. Thence, by abuse of notation, we associate $\emptyset$ in this case with the empty list $[]$. $\square$

Computing Fault Probabilities of Formulas.   Then, GETFORMULAPROBS is called in line 5 with the KB $\mathcal{K}$ and the function $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}} : \widetilde{\mathcal{K}} \cup \overline{\mathcal{K}} \to (0,1]$ as inputs. The function first applies Formula 4.2 to compute probabilities for each formula in $\mathcal{K}$, then applies Formula 4.7 to these probabilities leading to the output $p_{\mathcal{K}} : \mathcal{K} \to (0, 0.5)$, a function that assigns a value $p_{\mathcal{K}}(ax) \in (0, 0.5)$ to each $ax \in \mathcal{K}$.

Computing Leading Diagnoses.   At this point, all input arguments required by for the hitting set tree construction are instantiated. So, the algorithm enters the while loop in line 6. As a first step within the loop, either STATICHS, if $mode = static$, or DYNAMICHS, otherwise, is called in order to obtain a tuple including a set of leading diagnoses along with variables that store the "state" of the (partial) hitting set tree constructed so far and facilitate the reuse of this tree in the next iteration.

In concrete terms, STATICHS accepts the arguments $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, $\mathbf{Q}$, t, $n_{\min}$, $n_{\max}$, $\mathbf{C}_{calc}$, $\mathbf{D}_{\checkmark}$, $\mathbf{D}_{\times}$, $p_{\mathcal{K}}()$, P' and N' and returns a tuple $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times} \rangle$ the elements of which are defined as follows:

- $\mathbf{D}$ is the current set of leading diagnoses such that

  (a) $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ is the set of most probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that satisfy all test cases P' and N' such that

    (i) $n_{\min} \leq |\mathbf{D}| \leq n_{\max}$ and
    (ii) $\mathbf{D} \supset \mathbf{D}_{\checkmark}$,

    if such a set $\mathbf{D}$ exists; or

  (b) $\mathbf{D}$ is equal to the set of all minimal diagnoses $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$, otherwise;

  where "most-probable" refers to the diagnosis probability measure $p_{\mathbf{D}, prio}()$ obtained from $p_{\mathcal{K}}()$ by application of Formulas 4.3 and 4.4.

- $\mathbf{Q}$ is the current queue of open nodes of the hitting set tree.

- $\mathbf{C}_{calc} \subseteq \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ is the set of all computed minimal conflict sets w.r.t. the input DPI throughout all calls of STATICHS during the execution of Algorithm 5 so far.

- $\mathbf{D}_{\times}$ comprises all computed minimal diagnoses throughout all calls of STATICHS during the execution of Algorithm 5 so far where each $\mathcal{D} \in \mathbf{D}_{\times}$ has been invalidated by some test case in P' or N'.

Similarly, DYNAMICHS accepts the arguments $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, $\mathbf{Q}$, $\mathbf{Q}_{dup}$, t, $n_{\min}$, $n_{\max}$, $\mathbf{C}_{calc}$, $\mathbf{D}_{\checkmark}$, $\mathbf{D}_{\times}$, $p_{\mathcal{K}}()$, P', N' and $\mathbf{D}_{\supset}$ and returns a tuple $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times}, \mathbf{D}_{\supset}, \mathbf{Q}_{dup} \rangle$ the elements of which are defined as follows:

- $\mathbf{D}$ is the current set of leading diagnoses such that

  (a) $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ is the set of most probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ such that

    (i) $n_{\min} \leq |\mathbf{D}| \leq n_{\max}$ and

      (ii) $\mathbf{D} \setminus \mathbf{D}_{\checkmark} \neq \emptyset$,

      if such a set $\mathbf{D}$ exists, or

    (b) $\mathbf{D}$ is equal to the set of all minimal diagnoses $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$, otherwise,

where "most-probable" refers to the diagnosis probability measure $p_{\mathbf{D},prio}()$ obtained from $p_{\mathcal{K}}()$ by application of Formulas 4.3 and 4.4.

- $\mathbf{Q}$ is the current queue of open (non-labeled) nodes of the hitting set tree,

- $\mathbf{C}_{calc}$ is a set of conflict sets w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$,

- $\mathbf{D}_{\times} = \emptyset$,

- $\mathbf{D}_{\supset}$ is the set of all processed nodes so far throughout the execution of Algorithm 5 that are non-minimal diagnoses w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ and

- $\mathbf{Q}_{dup}$ includes all duplicate nodes found so far throughout the execution of Algorithm 5 (for a detailed explanation see Chapter 12 and Algorithm 8).

Remark 9.5 It is very important to notice that the function $p_{nodes}()$ for $p() := p_{\mathcal{K}}()$ as specified by Definition 4.9 on page 73 imposes the same order on a set of minimal diagnoses as the a-priori probability measure $p_{\mathbf{D},prio}()$. That is $p_{nodes}(\mathcal{D}) = c \cdot p_{\mathbf{D},prio}(\mathcal{D})$ for all minimal diagnoses $\mathcal{D}$ w.r.t. a DPI where c is a constant (which is the same for all diagnoses $\mathcal{D}$). The difference between both functions is that $p_{nodes}()$ is defined for all $X \subseteq \mathcal{K}$ whereas $p_{\mathbf{D},prio}()$ is only defined for (leading) minimal diagnoses $\mathcal{D} \subseteq \mathcal{K}$. Further on $p_{\mathbf{D},prio}()$ is normalized whereas $p_{nodes}()$ is not which accounts for the (normalization) constant c. The function $p_{nodes}()$ is essential for the best-first construction of the hitting set tree in STATICHS and DYNAMICHS since it allows for the assignment of a "probability" to non-diagnoses (cf. the discussion of Formula 4.6 on page 73). Since the input argument $p()$ (which is the same for all calls) to STATICHS as well as DYNAMICHS is equal to $p_{\mathcal{K}}()$ by lines 8 and 10 in Algorithm 5, the set $\mathbf{D}$ returned by STATICHS (DYNAMICHS) is also the set of most probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ ($\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$) *as per the function* $p_{nodes}()$ (cf. Proposition 11.1 and Corollary 12.8). $\quad\square$

Remark 9.6 Notice that the return parameter that is relevant for the main purpose of Algorithm 5, namely to compute a query and thereby obtain a new test case classified by the user, is solely the set of leading diagnoses $\mathbf{D}$. The other return parameters serve as a means to store the state of the hitting set tree that is gradually built up by successive calls of STATICHS (if $mode = static$) and DYNAMICHS (if $mode = dynamic$), respectively. Whereas $\mathbf{Q}$ and $\mathbf{C}_{calc}$ (and $\mathbf{D}_{\supset}$ and $\mathbf{Q}_{dup}$ in case of DYNAMICHS) are never modified until the next call to STATICHS or DYNAMICHS, the sets $\mathbf{D}_{\checkmark}$ and $\mathbf{D}_{\times}$ are only changed once, after the subset of invalidated leading diagnoses $\mathbf{D}_{out}$ is known, in lines 21 and 22. $\quad\square$

At this moment, we do not go into detail regarding the way how leading diagnoses are computed by STATICHS and DYNAMICHS. We simply suppose that both functions act in a manner that the outputs just specified are returned for the given inputs. An in-depth delineation of both functions will be given in Chapters 11 and 12 in Part III. Further note that the return parameter $\mathbf{D}$ is stored in variable $\mathbf{D}_{\checkmark}$ from line 10 on.

Computing a Probability Distribution of Leading Diagnoses. After the set of leading diagnoses $\mathbf{D}_{\checkmark}$ has been computed, the variables $\mathbf{D}_{\checkmark}$, $p_{\mathcal{K}}()$, $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and QA are used as arguments to the function GETPROBDIST (see Algorithm 6) which computes a probability distribution of the leading diagnoses, i.e. a probability measure $p_{\mathbf{D}}()$ for the probability space with sample space $\Omega = \mathbf{D}_{\checkmark}$ (cf. Section 4.6). As a first action to achieve this, the (a-priori) probabilities $p_{\mathbf{D},prio}(\mathcal{D})$ for $\mathcal{D} \in \mathbf{D}_{\checkmark}$ are computed from

the (a-priori) probabilities $p_{\mathcal{K}}(ax)$ for formulas $ax \in \mathcal{K}$ as per Formula 4.3 (GETPRIODIAGPROBS in line 29). Application of Formula 4.4 is not necessary at this point as probabilities are anyhow normalized at the end of GETPROBDIST (line 44). Notice that the function $p_{\mathcal{K}}()$ remains constant, i.e. unmodified, throughout the entire execution of Algorithm 5.

Now, since a-priori diagnosis probabilities assigned by $p_{\mathbf{D},prio}()$ directly rely upon $p_{\mathcal{K}}()$ which in turn is computed directly from the initially given fault probabilities $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}()$, the probability measure $p_{\mathbf{D},prio}()$ is adapted to yield *a-posteriori diagnosis probabilities* $p_{\mathbf{D}}()$ in order to reflect the new evidence provided by the collected test cases P' and N'.

The a-posteriori probability of a current leading diagnosis $\mathcal{D}$ in $\mathbf{D}_{\checkmark}$ is $p_{\mathbf{D}}(\mathcal{D} \,|\, QA)$ and can be computed by means of Bayes' Theorem (Formula 4.5) from $p_{\mathbf{D},prio}()$ as follows.

$$p_{\mathbf{D}}(\mathcal{D} \,|\, QA) = \frac{p_{\mathbf{D},prio}(QA \,|\, \mathcal{D}) \; p_{\mathbf{D},prio}(\mathcal{D})}{p_{\mathbf{D},prio}(QA)}$$

where QA is the chronologically ordered list of queries and user answers collected so far during the execution of Algorithm 5 (see page 127). We point out that $p_{\mathbf{D},prio}(QA)$ is only a normalization factor that is equal for each diagnosis and thus does not need to be explicitly computed. The crucial factor is

$$p_{\mathbf{D},prio}(QA \,|\, \mathcal{D}) = p_{\mathbf{D},prio}(\forall \, \langle Q, u(Q) \rangle \in QA : Q = u(Q) \,|\, \mathcal{D})$$

which describes the probability of getting exactly the answer $u(Q)$ for each query $Q \in P' \cup N'$ under the assumption that $\mathcal{D}$ corresponds to the true diagnosis $\mathcal{D}_t$, i.e. $\mathcal{D}_t = \mathcal{D}$. In other words, $p_{\mathbf{D},prio}(QA \,|\, \mathcal{D})$ is the probability of QA under the assumption that the user answers in a way that $u(Q) = true$ if $\mathcal{D} \in \mathbf{D}^+(Q)$ and $u(Q) = false$ if $\mathcal{D} \in \mathbf{D}^-(Q)$.

For a single query Q_i, the probability $p_{\mathbf{D},prio}(Q_i = u(Q_i) \,|\, \mathcal{D})$ is defined as (cf. [dKW87])

$$p_{\mathbf{D},prio}(Q_i = u(Q_i) \,|\, \mathcal{D}) = \begin{cases} 1, & \text{if } \mathcal{D} \in \mathbf{D}^+(Q_i) \\ 0, & \text{if } \mathcal{D} \in \mathbf{D}^-(Q_i) \\ \frac{1}{2}, & \text{if } \mathcal{D} \in \mathbf{D}^0(Q_i) \end{cases} \qquad (9.1)$$

for $u(Q_i) = true$ and

$$p_{\mathbf{D},prio}(Q_i = u(Q_i) \,|\, \mathcal{D}) = \begin{cases} 1, & \text{if } \mathcal{D} \in \mathbf{D}^-(Q_i) \\ 0, & \text{if } \mathcal{D} \in \mathbf{D}^+(Q_i) \\ \frac{1}{2}, & \text{if } \mathcal{D} \in \mathbf{D}^0(Q_i) \end{cases} \qquad (9.2)$$

for $u(Q_i) = false$ where $\mathbf{D}^+(Q_i)$, $\mathbf{D}^-(Q_i)$ and $\mathbf{D}^0(Q_i)$ are computed w.r.t. the DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P'', N \cup N'' \rangle$ where P'' and N'', respectively, include all test cases collected *prior* to Q_i, i.e. $P'' \cup N'' = \{Q_1, \ldots, Q_{i-1}\}$ if queries are numbered chronologically. That is, if $\mathcal{D}$ predicted the answer $u(Q_i)$ to Q_i given by the user, the probability is 1, zero if $\mathcal{D}$ predicted the converse answer $\neg u(Q_i)$ and $\frac{1}{2}$ if $\mathcal{D}$ did not predict any answer to Q_i.

So, aside from the normalization factor (see above), $p_{\mathbf{D},prio}(Q_i = u(Q_i) \,|\, \mathcal{D})$ is the factor by which the a-priori probability $p_{\mathbf{D},prio}(\mathcal{D})$ must be multiplied to obtain the a-posteriori probability $p_{\mathbf{D}}(\mathcal{D})$ of a diagnosis $\mathcal{D}$ after a single query Q_i has been answered and added as a test case to the DPI.

The intuitive explanation for the update by this factor is that if $\mathcal{D}$ predicted (at least) one answer $u(Q)$ conversely as given by the user, then $\mathcal{D}$ is a-posteriori impossible since it has already been invalidated by the addition of test case Q. In case a diagnosis has never predicted the wrong answer, but did not predict any answer for many queries so far, then it is a-posteriori more unlikely than a diagnosis that did predict a correct answer more often. That is, our a-posteriori degree of belief that $\mathcal{D}$ is the correct diagnosis is

the higher, the more often $\mathcal{D}$ had predicted answers to queries that were later actually given by the user (cf. Section 7.4 for an explanation what we mean by "predict").

The value of $p_{\mathbf{D},prio}(Q_i = u(Q_i) \,|\, \mathcal{D})$ can be computed by use of QA and the q-partitions $\mathfrak{P}(Q_1)$, $\dots$, $\mathfrak{P}(Q_{i-1})$ of the *current* set of leading diagnoses $\mathbf{D}_{\checkmark}$ (for which a-posteriori probabilities are to be computed) for all queries $Q_1, \dots, Q_{i-1}$ answered before query Q_i. Thereby, each $\mathfrak{P}(Q_j)$ where $j \in \{1, \dots, i-1\}$ must be computed for a DPI where only $Q_1, \dots, Q_{j-1}$ are incorporated as test cases.

Taking these thoughts into account, GETPROBDIST (Algorithm 6) updates $p_{\mathbf{D},prio}(\mathcal{D})$ for each diagnosis $\mathcal{D} \in \mathbf{D}_{\checkmark}$ in that it runs through all query-answer pairs $\langle Q, u(Q) \rangle$ in QA chronologically and for each $\mathcal{D} \in \mathbf{D}_{\checkmark}$ it multiplies $p_{\mathbf{D},prio}(\mathcal{D})$ by $\frac{1}{2}$ if $\mathcal{D} \in \mathbf{D}^0(Q)$ as per Formulas 9.1 and 9.2. For each check whether a diagnosis is in $\mathbf{D}^0(Q)$ in lines 34 and 39 a DPI is used that already incorporates all test cases P'' and N'' that have been added chronologically before Q was asked. This is achieved by updating P'' and N'' successively (lines 36 and 41). After all elements of QA have been processed, the updated diagnosis probabilities are finally normalized (line 44, cf. Formula 4.4 on page 72) and the resulting function $p_{\mathbf{D},prio}()$ is returned.

Remark 9.7 Note that the function GETPROBDIST exploits the fact that all diagnoses in $\mathcal{D}_{\checkmark}$ are leading diagnoses w.r.t. the *current* DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ which guarantees that none of these diagnoses has been invalidated by any of the test cases in P' or in N' added throughout the execution of Algorithm 5 (cf. Proposition 12.3 given later). Hence, it is clear that each $\mathcal{D} \in \mathbf{D}_{\checkmark}$ must be in $\mathbf{D}^+(Q) \cup \mathbf{D}^0(Q)$ if $u(Q) = true$ and in $\mathbf{D}^-(Q) \cup \mathbf{D}^0(Q)$ if $u(Q) = false$, and it is only tested whether $\mathcal{D} \notin \mathbf{D}^+(Q)$ in the prior case (line 34) and whether $\mathcal{D} \notin \mathbf{D}^-(Q)$ in the latter (line 39). It must be further noted that, in case of $mode = dynamic$, diagnoses in $\mathbf{D}_{\checkmark}$ are not necessarily minimal diagnoses w.r.t. the intermediate DPIs $\langle \mathcal{K}, \mathcal{B}, P \cup P'', N \cup N'' \rangle$ that are used for the probability update. However, this is not problematic since any set of (minimal and/or non-minimal) diagnoses is partitioned into the three sets $\mathbf{D}^+(Q)$, $\mathbf{D}^-(Q)$ and $\mathbf{D}^0(Q)$ by a query Q (cf. Remark 7.3) wherefore $\mathfrak{P}(Q)$ exists for any set $\mathbf{D}_{\checkmark}$. Thence, the correctness of GETPROBDIST remains unaffected by the usage of the setting $mode = dynamic$. □

Remark 9.8 We want to emphasize that an adaptation of $p_{\mathbf{D},prio}(\mathcal{D})$ is only necessary in case $\mathcal{D} \in \mathbf{D}^0(Q_j)$ for some query Q_j answered so far during the execution of Algorithm 5 as otherwise a multiplication by 1 is required which does not change $p_{\mathbf{D},prio}(\mathcal{D})$.

For the case of static debugging ($mode = static$), an immediate implication of this is the following: The restriction of asking the user *only* queries Q_j w.r.t. a DPI with the property that *no minimal diagnosis* w.r.t. this DPI can be an element of $\mathbf{D}^0(Q_j)$ makes the probability update for each diagnosis in $\mathbf{D}_{\checkmark}$ equivalent to a multiplication by 1 and hence obsolete. This must be the case since each diagnosis in $\mathbf{D}_{\checkmark}$ which is a subset of $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ (see Section 9.2.2) must be a *minimal* diagnosis w.r.t. each intermediate DPI (which includes a superset of the test cases in the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and a subset of the test cases in the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$) as will be substantiated by Proposition 12.5 given later. Consequently, such a scenario implicates that the order of diagnoses computed by STATICHS corresponds to the best-first order also w.r.t. the a-posteriori diagnosis probabilities (cf. Remark 9.3).

The approach of only using queries with this property is feasible, e.g. by using a GETENTAILMENTS function in conformity with Proposition 8.3 for the generation of the query pool (GETPOOLOFQUERIES). Such a type of queries is also favorable from the discrimination point of view, as we pointed out in Section 8.2. An improvement of static debugging with this type of queries is to deactivate the probability update, i.e. replace line 11 in Algorithm 5 by line 29 of Algorithm 6. This improvement is not shown in Algorithm 5.

In a dynamic debugging session ($mode = dynamic$), on the contrary, the usage of such queries does not guarantee the triviality of the probability update. For, also if no minimal diagnosis w.r.t. the DPI (for which a query Q_j is computed) can be an element of $\mathbf{D}^0(Q_j)$, there may be some non-minimal

one which is. For example, for any admissible DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is holds that $\mathcal{D} := \mathcal{K}$ is a diagnosis (cf. Proposition 3.4 and Definition 3.6), albeit in most cases a non-minimal one. In such a case, $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup U_P$ which is equal to $\mathcal{B} \cup U_P$ cannot entail Q_j. Because, were this the case, then all minimal diagnoses $\mathcal{D}_i \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ would be elements of $\mathbf{D}^+(Q_j)$ as each $\mathcal{K}_i^* \supseteq \mathcal{B} \cup U_P$ and thus each $\mathcal{K}_i^* \models Q_j$ by the monotonicity of $\mathcal{L}$. Hence, this would be a contradiction to the fact that Q_j is a query w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ by Corollary 7.2. On the other hand, $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup U_P \cup Q_j = \mathcal{B} \cup U_P \cup Q_j$ cannot violate any $x \in N \cup R$. Since, if this were the case, then adding Q_j to the positive test cases would lead to a non-admissible DPI $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q_j\}, N \rangle_R$. By Corollary 7.3, this would be a contradiction to the fact that Q_j is a query w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Thence, $\mathcal{D} \in \mathbf{D}^0(Q_j)$ must hold for the assumed non-minimal diagnosis $\mathcal{D}$. From that we conclude that the probability update in dynamic debugging cannot be made obsolete in general by the usage of such a type of queries. $\qquad \square$

Stop Criterion and Output. The (a-posteriori) probability distribution $p_{\mathbf{D}}()$ of leading diagnoses $\mathbf{D}_{\checkmark}$ is then used in line 12 of Algorithm 5 to compute the mode of this distribution, i.e. the one diagnosis $\mathcal{D}_{\max} \in \mathbf{D}_{\checkmark}$ with maximum probability according to $p_{\mathbf{D}}()$.

In the sequel, $\mathcal{D}_{\max}$ is used to check the stop criterion (line 13), namely whether $\mathcal{D}_{\max}$ has a probability greater than or equal to $1 - \sigma$. If this is the case and $mode = static$, the function GETSOLKB computes a maximal solution KB w.r.t. the input DPI as $(\mathcal{K} \setminus \mathcal{D}_{\max}) \cup U_P$ by means of the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$, P' and $\mathcal{D}_{\max}$. Given that $mode = dynamic$, GETSOLKB returns a maximal solution KB w.r.t. the current DPI as $(\mathcal{K} \setminus \mathcal{D}_{\max}) \cup U_{P \cup P'}$ by means of the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ and $\mathcal{D}_{\max}$. This solution KB is then returned as an output of Algorithm 5. If, on the other hand, the stop criterion is not met, the algorithm continues the execution with the computation of another query.

Remark 9.9 Notice that the returned maximal solution KB $(\mathcal{K} \setminus \mathcal{D}_{\max}) \cup U_P$ w.r.t. the input DPI in case $mode = static$ can be easily extended to constitute a maximal solution KB w.r.t. the current DPI, namely by extending it by $U_{P'}$. If $mode = dynamic$, then the KB output in line 14 is a maximal solution KB w.r.t. *the current DPI*, but possibly a non-maximal solution KB w.r.t. the input DPI. $\qquad \square$

Query Computation and User Interaction. In line 16, the function CALCQUERY is applied to compute a query and the associated q-partition by means of the leading diagnoses $\mathbf{D}_{\checkmark}$, (possibly) the collected data $qData$, the probability distribution $p_{\mathbf{D}}()$ of the leading diagnoses, a query selection function $qsm()$ (which might exploit the function $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}()$), a parameter q determining the size of the computed query pool and the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$.

As a first step within CALCQUERY, the function GETPOOLOFQUERIES computes a query pool $\mathbf{QP}$ as detailed in Chapter 8 from $\mathbf{D}_{\checkmark}$, q and $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. Then, the best tuple $\langle Q, \mathfrak{P}(Q) \rangle \in \mathbf{QP}$ according to the function $qsm()$ is searched for and finally returned as the output of CALCQUERY. During the query selection process, the evaluation of the query selection measure $qsm(Q) \in \mathbb{R}$ for queries Q where $\langle Q, \mathfrak{P}(Q) \rangle \in \mathbf{QP}$ may require $qData$, the fault probabilities $p_{\mathbf{D}}()$ of leading diagnoses as well as the fault probabilities $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}()$ of syntactical elements in $\mathcal{K}$. This depends on which concrete measure $qsm()$ is employed (see Section 9.3 which presents some possible measures).

As a next step, the query Q of the best tuple $\langle Q, \mathfrak{P}(Q) \rangle \in \mathbf{QP}$ is presented to the interacting user in line 17 which is the only place in Algorithm 5 where user interaction takes place. The user is modeled as a *deterministic* function $u : \mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle} \to \{true, false\}$ that allocates a positive ($true$) or negative ($false$) answer to each query w.r.t. any set of leading diagnoses $\mathbf{D}$ for some current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle$. The answer $u(Q)$ given by the user is stored in the variable $answer$.

Remark 9.10 We want to point out that the algorithm can be easily adapted to allow a user to reject

queries, e.g. if they are not sure how to answer. That is, the user function might be modeled as $u :$ $\mathbf{Q}_{\mathbf{D},\langle \mathcal{K},\mathcal{B},P\cup P',N\cup N'\rangle} \to \{true, false, unknown\}$ where $u(Q) = unknown$ signifies the rejection of query Q. In this case, an accordingly modified version of Algorithm 5 would calculate an alternative query w.r.t. $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N'\rangle$, e.g. the second best one according to the query selection measure $qsm()$ among all tuples in $\mathbf{QP}$ (this potential feature is not shown in Algorithm 5). In this vein, a total of $|\mathbf{QP}| - 1$ queries can be dismissed per set of leading diagnoses $\mathbf{D}$.

We want to accentuate that the presented interactive algorithm might be easily adapted to cope with queries whose answer is unknown to the user, but a definite assumption for the algorithm to return a correct solution is a user that does not give wrong answers. In other words, the algorithm does not provide inherent mechanisms that allow for the detection of wrong answers or for the debugging of the KB debugging procedure (keyword "garbage in, garbage out"). So, we suppose the function $u()$ to be *deterministic* which prohibits the situation that a user might change their mind at a later point in time. Of course, this is still a possible scenario in practice, but in case it arises, a user has to revise, i.e. delete or edit, specified test cases they disagree with by hand before a new debugging session using the modified DPI might be started.

Another remark at this place concerns the way a user might choose to answer the query. A "minimal" feedback of a user that we regard as an answer to a query Q is to merely say $true$, i.e. each formula in Q (or the conjunction of formulas in Q) must be entailed by the correct KB, or $false$, i.e. at least one formula in Q (or the conjunction of formulas in Q) must not be entailed by the correct KB. The presented algorithm (Algorithm 5) is designed to deal with exactly this kind of an answer. However, imagine a user being presented Q and think of how they might proceed in order to come up with an answer to Q. The first observation is that, in order to respond by $true$, a user must definitely scrutinize each single formula in Q because otherwise they could never decide for sure whether the conjunction of all formulas in Q is correct. Another observation is that a user might cease to go through the rest of the formulas in case they have already identified one that must not be an entailment of the desired KB. For, in this situation, the overall query Q is already $false$. This however indicates that at least one formula must be known to be correct or false whatever answer is given to Q. Therefore, we can usually expect a user to be able to give exactly this information, namely one formula in Q that must be incorrect, additionally to answering by $false$. This extra piece of information can be exploited to achieve better space and time efficiency in the context of diagnosis computation. Proposing more efficient algorithms that exploit this information is a topic for future work. □

Incorporating the New Information. The new information represented by the answer $answer$ to Q is incorporated (lines 18-26) by updating values of all relevant parameters. First, by means of the function APPEND, the tuple consisting of the answered query Q and the corresponding answer $answer$ given by the user is added as a last element to the chronological list of queries and answers QA that is used for the next probability update (line 11).

Then, the subset $\mathbf{D}_{out}$ of the leading diagnoses $\mathbf{D}_{\checkmark}$ that gets invalidated after adding Q to the positive or negative test cases of the DPI, respectively, is computed by the function GETINVALIDDIAGS that gets the q-partition $\mathfrak{P}(Q) = \langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q)\rangle$ of Q and $answer$ as input arguments. $\mathbf{D}_{out}$ then corresponds to the set $\mathbf{D}^-(Q)$ given that $answer$ is $true$ and to $\mathbf{D}^+(Q)$ otherwise (cf. Section 7.4). Note that $\emptyset \subset \mathbf{D}_{out} \subset \mathbf{D}_{\checkmark}$ holds by Proposition 7.4 and since Q is a query w.r.t. $\mathbf{D}_{\checkmark}$ (since $\mathbf{D}_{\checkmark}$ is given as an input to CALCQUERY).

As a next step, the data $qData$ is updated. As already pointed out in Section 9.2.3, the form of the variable $qData$ depends on the employed query selection measure $qsm()$ and so do the actions that are performed by UPDATEQDATA.

In order to communicate the impact of the answered query to the hitting set tree algorithm (either STATICHS or DYNAMICHS), the set of invalidated leading diagnoses $\mathbf{D}_{out}$ is deleted from the leading

diagnoses $\mathbf{D}_{\checkmark}$ and added to $\mathbf{D}_{\times}$. After this update, $\mathbf{D}_{\checkmark}$ includes all diagnoses that have been computed by the hitting set tree algorithm so far that are minimal diagnoses w.r.t. the current DPI.

Finally, the new test case Q is added to the new positive test cases P' if $answer$ is $true$ and to the new negative test cases N' in case of $answer = false$.

9.3 Query Selection Measures

In this section, we give a brief introduction to some query selection measures $qsm()$ that have been suggested and evaluated in literature within the scope of KB or ontology debugging [SFFR12, RSFF13]. Such query selection measures, when used as a parameter in an interactive KB debugging algorithm such as the one described by Algorithm 5, aim at solving the following optimization problems. In Interactive Dynamic KB Debugging, the problem is defined as follows:

> **Problem Definition 9.1.** *The task is to solve the problem specified by Problem Definition 6.1 in a way that $|P'| + |N'|$ is minimal.*

In Interactive Static KB Debugging, the problem is defined as follows:

> **Problem Definition 9.2.** *The task is to solve the problem specified by Problem Definition 6.2 in a way that $|P'| + |N'|$ is minimal.*

That is, these optimization problems aim at the minimization of user effort during interactive KB debugging. In other words, the goal is the minimization of the number of queries required to be asked to a user in order to solve the Interactive Static KB Debugging or the Interactive Dynamic KB Debugging Problem, respectively.

In our previous work [SFFR12], we have discussed entropy-based ($\mathsf{ENT}()$) and split-in-half ($\mathsf{SPL}()$) query selection measures.

Entropy-Based Query Selection.   A best query Q_{ENT} according to $\mathsf{ENT}()$ has a maximal information gain among all queries Q where $\langle Q, \mathfrak{P}(Q) \rangle \in \mathbf{QP}$. In other words, Q_{ENT} minimizes the expected entropy of the probability distribution of the leading diagnoses $\mathbf{D}_{\checkmark}$ after Q_{ENT} has been added as a test case to the DPI based on the user's answer $u(Q_{\mathsf{ENT}})$. As shown in [dKW87], this leads to the definition

$$\mathsf{ENT}(Q) := \sum_{a \in \{true, false\}} p(Q = a) \log p(Q = a) + p(\mathbf{D}^0(Q))$$

where $p()$ in the case of our algorithm corresponds to the leading diagnoses probability measure $p_{\mathbf{D}}()$ computed in line 11 in Algorithm 5 and

$$p(Q = true) = p(\mathbf{D}^+(Q)) + \frac{1}{2} p(\mathbf{D}^0(Q))$$

$$p(Q = false) = p(\mathbf{D}^-(Q)) + \frac{1}{2} p(\mathbf{D}^0(Q))$$

(cf. Section 7.4) where

$$p(\mathbf{D}^+(Q)) = \sum_{\mathcal{D} \in \mathbf{D}^+(Q)} p(\mathcal{D})$$

$$p(\mathbf{D}^-(Q)) = \sum_{\mathcal{D} \in \mathbf{D}^-(Q)} p(\mathcal{D})$$

$$p(\mathbf{D}^0(Q)) = \sum_{\mathcal{D} \in \mathbf{D}^0(Q)} p(\mathcal{D})$$

Then, the best query in a pool $\mathbf{QP}$ according to $qsm() := \mathsf{ENT}()$ is

$$Q_{\mathsf{ENT}} = \underset{\{Q \mid \langle Q, \mathfrak{P}(Q) \rangle \in \mathbf{QP}\}}{\arg\min} \mathsf{ENT}(Q)$$

So, theoretically optimal w.r.t. $\mathsf{ENT}()$ is a query Q whose positive and negative answers are equally likely and for which $\mathbf{D}^0(Q)$ is the empty set. In other words, the best query has the property that the *sum of probabilities* of leading diagnoses predicting the positive answer as well as the *sum of probabilities* of leading diagnoses predicting the negative answer is 50%.

Split-In-Half Query Selection. For the selection criterion $qsm() := \mathsf{SPL}()$, on the other hand, the query

$$Q_{\mathsf{SPL}} = \underset{\{Q \mid \langle Q, \mathfrak{P}(Q) \rangle \in \mathbf{QP}\}}{\arg\min} \mathsf{SPL}(Q)$$

is preferred where

$$\mathsf{SPL}(Q) := \big| \, |\mathbf{D}^+(Q)| - |\mathbf{D}^-(Q)| \, \big| + |\mathbf{D}^0(Q)|$$

Hence, this measure is optimized by queries Q for which the *number* of leading diagnoses predicting the positive answer is equal to the *number* of leading diagnoses predicting the negative answer and for which $\mathbf{D}^0(Q)$ is the empty set.

Risk-Optimized Query Selection. For scenarios where a-priori probabilities are vague, we have presented another more complex query selection measure $\mathsf{RIO}()$ in [RSFF13] which uses a reinforcement learning strategy to constantly adapt some "risk" parameter that indicates the current amount of trust in the probabilities. Whereas $\mathsf{ENT}()$ and $\mathsf{SPL}()$ do not rely on $qData$, this learning strategy does so and requires the invalidation rate or "performance", i.e. $\frac{|\mathbf{D}_{out}|}{|\mathbf{D}_{\checkmark}|}$, of the previous iteration for the adaptation of the learning parameter. As long as the invalidation rate is "good", the trust in the current (a-posteriori) probabilities – that strongly depend on the vague a-priori probabilities – is high, but it is gradually decreased after observing "worse" performance, and so on. High trust in the probabilities means usage of $\mathsf{ENT}()$ which can exploit high quality fault information well as demonstrated in the experiments conducted in [SFFR12], whereas low trust involves selection of queries that guarantee a higher worst case invalidation rate, i.e. have similar properties to queries $\mathsf{SPL}()$ would select.

Example 9.1 Let us reconsider the queries and associated q-partitions for the example DPI of Table 15.2 that are depicted by Table 8.3 on page 113. Let us denote by $Q_i \prec_M Q_j$ that Q_i is preferred over Q_j and by $Q_i \prec\succ_M Q_j$ that Q_i is equally preferable as Q_j if the query selection measure $qsm() := M$ is used. Furthermore, we make the assumption that the probability distribution $p_{\mathbf{D}}$ of the (leading) diagnoses $\mathbf{D}_{\checkmark} = \{\mathcal{D}_1, \ldots, \mathcal{D}_4\}$ is as shown in Table 9.1.

Then, we make the following observations:

- Q_6 is the theoretically optimal query w.r.t. $\mathsf{ENT}()$ since $p_{\mathbf{D}}(\mathbf{D}^+(Q_6)) = 0.5$, $p_{\mathbf{D}}(\mathbf{D}^-(Q_6)) = 0.5$ and $\mathbf{D}^0(Q_6) = \emptyset$, i.e. the positive and the negative answer have equal probabilities of 50% and thus Q_6 the highest theoretically possible information gain of 1 (bit). This can be compared with one toss of a coin where the information gain of tossing the coin and checking whether it is head or tail is highest in a case where the coin is fair. For a coin that shows head with a probability of 0.95, conversely, the information gain of tossing the coin is rather small since we are already quite sure about the result in advance.

$\mathcal{D} \in \mathbf{D}_{\checkmark}$	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$
$p_{\mathbf{D}}(\mathcal{D})$	0.15	0.3	0.05	0.5

Table 9.1: (Example 9.1) Diagnosis probabilities for the example DPI given by Table 15.2.

- $Q_9 \prec_M Q_5$ as well as $Q_9 \prec_M Q_2$ for $M \in \{\mathsf{SPL}(), \mathsf{ENT}()\}$ because both Q_5 and Q_2 share one set in $\{\mathbf{D}^+, \mathbf{D}^-\}$ with Q_9, but exhibit a non-empty set $\mathbf{D}^0$ whereas $\mathbf{D}^0(Q_9) = \emptyset$. This shows that both split-in-half and entropy-based query selection penalize a query Q if there are leading diagnoses that are definitely not discriminated by it, i.e. $\mathbf{D}^0(Q) \neq \emptyset$. This is perfectly desirable as we discussed.

- $Q_4 \prec\succ_M Q_{10}$ for $M \in \{\mathsf{SPL}(), \mathsf{ENT}()\}$ since their q-partitions differ just by commutation of the sets $\mathbf{D}^+$ and $\mathbf{D}^-$. This is what one would expect of such a measure, i.e. that it does not matter whether the positive or negative answer is more probable if the probability values are the same (in case of $\mathsf{ENT}()$) and whether the number of diagnoses predicting the positive or negative answer is higher if the numbers are the same (in case of $\mathsf{SPL}()$). However, notice that Q_4 might be much easier to comprehend and answer for the interacting user. Therefore, Q_4 might be preferred in a scenario where some second measure $qsm_2()$ comes into play to identify a best query among equally preferable queries w.r.t. some $qsm_1()$ that is used as a primary measure. For, example some "query-easiness" measure $qsm_2()$ might be employed after $qsm_1() \in \{\mathsf{SPL}(), \mathsf{ENT}()\}$ has filtered out an equally preferable set of queries; in this case let this set be $\{Q_4, Q_{10}\}$. The measure $qsm_2()$ could be defined to simply count the logical connectives and quantifiers occurring in a query Q and pick one for which this number is minimal. In this case, this number would be 0 for Q_4 and 7 for Q_{10}, wherefore Q_4 would be decisively better than Q_{10} w.r.t. $qsm_2()$.

- It holds that $Q_3 \prec_{\mathsf{ENT}()} Q_{10} \prec_{\mathsf{ENT}()} Q_1$, but $Q_3 \prec\succ_{\mathsf{SPL}()} Q_{10} \prec\succ_{\mathsf{SPL}()} Q_1$. The former holds since all three queries feature an empty set $\mathbf{D}^0$, but the difference between $p(\mathbf{D}^+)$ and $p(\mathbf{D}^-)$ is largest for Q_1 ($p(\mathbf{D}^+(Q_1)) = 0.95$), second largest for Q_{10} ($p(\mathbf{D}^-(Q_{10})) = 0.85$) and smallest for Q_3 ($p(\mathbf{D}^+(Q_3)) = 0.7$).

- Q_9 is the second best query among those given in Table 8.3 because both answers of it are almost equally probable (positive answer has a probability of 0.55 and negative answer a probability of 0.45).

- Queries Q_7, Q_8 and Q_9 are theoretically optimal w.r.t. the $\mathsf{SPL}()$ measure, since $\mathbf{D}^0 = \emptyset$ and $|\mathbf{D}^+| = |\mathbf{D}^-|$ for all of them.

- Regarding the $\mathsf{RIO}()$ measure, queries Q_7, Q_8 and Q_9 are "no risk" queries since they feature the maximum possible worst case elimination rate of 50%. Q_2 and Q_6, for instance, have a "higher risk" as their minimal invalidation rate amounts to only 25%. That is, if Q_2 (Q_6) is answered positively (negatively), then only one of four leading diagnoses is invalidated. $\qquad\square$

9.4   Interactive Debugging Algorithm: Correctness and Complexity

First, we prove the correctness of Proposition 9.1 on page 124 by using the results of Sections 11.4 and 12.4.10 which provide evidence for the correctness (soundness, completeness and optimality) of methods STATICHS and DYNAMICHS:

Proof of Proposition 9.1. First, we argue why Algorithm 5 must terminate. The function GETFORMU-LAPROBS in line 5 terminates since it applies Formulas 4.2 and 4.7 $|\mathcal{K}|$ times and $|\mathcal{K}|$ is finite by Definition 3.1. If $mode = static$, then STATICHS terminates due to Proposition 11.1. If $mode = dynamic$, then DYNAMICHS terminates due to Corollary 12.8. GETPROBDIST terminates since (1) the number of already answered queries $|QA|$ is finite, (2) $|\mathbf{D}_{\checkmark}|$ is finite since diagnoses are subsets of $\mathcal{K}$ and thus there is only a finite number of (minimal) diagnoses w.r.t. any DPI according to Definition 3.1 (since all sets included in the DPI are finite) and (3) reasoning (GETENTAILMENTS and ISKBVALID) is assumed to be decidable for the logic $\mathcal{L}$ over which the DPI is formulated as per Chapter 2. Further, GETMODE clearly terminates due to the fact that $|\mathbf{D}_{\checkmark}|$ is finite and returns the mode $\mathcal{D}_{\max}$ of the diagnoses probability distribution $p_{\mathbf{D}}()$ over the diagnoses in $\mathbf{D}_{\checkmark}$. Now, if the stop criterion $p_{\mathbf{D}}(\mathcal{D}_{\max}) \geq 1 - \sigma$ is met, then GETSOLKB is called. GETSOLKB simply deletes the given diagnosis $\mathcal{D}_{\max}$ from the given KB $\mathcal{K}$ and adds a finite set of formulas to it, and thence terminates.

If the stop criterion is not met, then $|\mathbf{D}_{\checkmark}| \geq 2$ must hold as otherwise the single diagnosis $\mathcal{D} \in \mathbf{D}_{\checkmark}$ would necessarily have fulfilled the stop criterion as its probability as per any probability measure over the sample space $\Omega := \mathbf{D}_{\checkmark}$ must be equal to 1 and thus greater than or equal to $1 - \sigma$ where $\sigma \geq 0$.

Due to $|\mathbf{D}_{\checkmark}| \geq 2$, Proposition 8.10 implies that GETPOOLOFQUERIES (called within CALCQUERY) terminates and yields a non-empty query pool as output. SELECTBESTQUERY (also called within CAL-CQUERY) terminates as well since it simply selects one query from the pool according to the measure $qsm()$ (cf. Section 9.3). Since we assume the interacting user to answer to a query or to reject it within finite time, $u(Q)$ also terminates. It is clear that APPEND terminates. GETINVALIDDIAGS simply extracts one entry of the given q-partition and thus terminates. Finally, UPDATEQDATA also terminates by assumption (no $qsm()$ must be used for which UPDATEQDATA might not terminate). As a consequence, all functions called in Algorithm 5 terminate. What remains to be proven is that the stop criterion must be met after a finite number of iterations, i.e. after a finite number of test cases have been added to the input DPI.

In $mode = static$ the stop criterion must be satisfied after a finite number of iterations due to the following argumentation:

- There is a finite set of minimal diagnoses w.r.t. the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ since each (minimal) diagnosis w.r.t. this DPI is a subset of $\mathcal{K}$ according to Definition 3.5 and since $|\mathcal{K}|$ is finite by Definition 3.1.

- In each iteration, one test case is added either to P' or N'.

- Each test case added to whatever set P' or N' invalidates at least one minimal diagnosis w.r.t. the input DPI in the set $\mathbf{D}_{\checkmark}$ by the definition of a query (Definition 7.1) and since each query is computed w.r.t. the leading diagnoses $\mathbf{D}_{\checkmark}$ by the correctness of GETPOOLOFQUERIES (cf. Proposition 8.10).

- $\mathbf{D}_{\checkmark}$ contains only minimal diagnoses w.r.t. the input DPI by Proposition 11.1.

- Also by Proposition 11.1, no invalidated minimal diagnosis w.r.t. the input DPI can be an element of some subsequent set of leading diagnoses $\mathbf{D}_{\checkmark}$.

- Therefore, unless the stop criterion is met before due to a sufficiently high probability of one of multiple leading diagnoses as per $p_{\mathbf{D}}()$, Algorithm 5 in $mode = static$ must arrive at a point where $|\mathbf{D}_{\checkmark}| = 1$ after a finite number of iterations. Note that $|\mathbf{D}_{\checkmark}| = 0$ is impossible due to the definition of a query (Definition 7.1) which ensures that each added test case leaves valid at least one minimal diagnosis in $\mathbf{D}_{\checkmark}$.

Algorithm 5 terminates in $mode = dynamic$ since for any sequence QA of queries that are added to the positive or negative test cases P' or N', respectively, there is a finite number k_{QA} such that there is no

more than one minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ for $|P'| + |N'| = k_{QA}$ wherefore the stop criterion must be met. Now, let us assume that the opposite holds. That is, there is a sequence QA^* of queries that are added to the positive or negative test cases P' or N', respectively, and for all natural numbers k there is more than one minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ for $|P'| + |N'| = k$. Then we argue as follows to derive a contradiction:

- There is a finite set of (minimal) diagnoses w.r.t. any DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ obtained from the input DPI by the addition of test cases. This is true since $|\mathcal{K}|$ is finite by Definition 3.1 and since each (minimal) diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ is a subset of $\mathcal{K}$ according to Definition 3.5.

- In each iteration, one test case is added either to P' or N'.

- Each test case added to whatever set P' or N' invalidates at least one minimal diagnosis w.r.t. the current DPI in the set $\mathbf{D}_{\checkmark}$ by the definition of a query (Definition 7.1) and since each query is computed w.r.t. the leading diagnoses $\mathbf{D}_{\checkmark}$ by the correctness of GETPOOLOFQUERIES (cf. Proposition 8.10).

- If DPI denotes the current DPI at the time DYNAMICHS is called, then the set $\mathbf{D}_{\checkmark}$ returned by DYNAMICHS is a subset of or equal to $\mathbf{mD}_{DPI}$, i.e. $\mathbf{D}_{\checkmark}$ contains only minimal diagnoses w.r.t. DPI by Corollary 12.8.

- Let $\langle DPI_0, DPI_1, \dots \rangle$ denote the sequence of DPIs encountered in the case of adding answered queries as test cases to the input DPI DPI_0 as per QA^*. Further, let $\langle \mathbf{aD}_0, \mathbf{aD}_1, \dots \rangle$ be the sequence such that $\mathbf{aD}_i := \mathbf{aD}_{DPI_i}, i = 0, 1, \dots$, i.e. $\mathbf{aD}_i$ is the set of all diagnoses w.r.t. DPI_i. Then $\mathbf{aD}_i \supset \mathbf{aD}_{i+1}$ for all $i \geq 0$ due to Corollary 12.4.

- As each query added as a test case to DPI_i leaves valid at least one (minimal) diagnosis w.r.t. DPI_i due to Definition 7.1, we have that $\mathbf{aD}_k \supset \emptyset$ for $k = 0, 1, \dots$.

- Since $\mathbf{aD}_i$ is finite, there must be some finite number k^* such that $|\mathbf{aD}_{k^*}| = 1$ wherefore $|\mathbf{mD}_{k^*}| = 1$ must also be valid. This is a contradiction.

Thence, Algorithm 5 terminates in any mode $mode$. Now, we show that propositions (1)-(6) of Proposition 9.1 hold for (i) $mode = static$ and (ii) $mode = dynamic$.

(i): First, by the proof so far, we have that Algorithm 5 in $mode = static$ given the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ terminates. Since the only point where the algorithm can terminate is line 14, GETSOLKB is called with arguments $\langle \mathcal{D}_{\max}, \langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R, P', static \rangle$. By the definition of GETSOLKB (see Section 9.2.4), we have that $(\mathcal{K} \setminus \mathcal{D}_{\max}) \cup U_P$ is returned by the algorithm.

Propositions (1) and (2) follow from the specification of the GETMODE function which is called with arguments $\langle \mathbf{D}_{\checkmark}, p_{\mathbf{D}}() \rangle$. Proposition (3) is true since GETSOLKB can never be reached without $p_{\mathbf{D}}(\mathcal{D}_{\max}) \geq 1 - \sigma$ being fulfilled. $\mathbf{D}_{\checkmark} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ is true due to Proposition 11.1, Remark 9.5 and the fact that $\mathbf{D}_{\checkmark}$ is obtained as an output of STATICHS. Hence, Proposition (4) holds. Proposition (5) is implied by Remark 9.5 and by the specification of the GETFORMULAPROBS function which computes $p_{\mathcal{K}}()$ from $p_{\overline{\mathcal{K}} \cup \overline{\mathcal{K}}}()$ as per Formulas 4.2 and 4.7 in line 5. Finally, Proposition (6) is a consequence of the definition of the GETPROBDIST function which accounts for the computation of $p_{\mathbf{D}}()$ from $p_{\mathcal{K}}()$, the input DPI, $\mathbf{D}_{\checkmark}$ and the chronological sequence of all queries and associated answers QA so far. Therefore, Proposition 9.1 is true for $mode = static$.

(ii): First, by the proof so far, we have that Algorithm 5 in $mode = dynamic$ given the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ terminates. Since the only point where the algorithm can terminate is line 14, GETSOLKB is called with arguments $\langle \mathcal{D}_{\max}, \langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R, P', dynamic \rangle$. By the definition of GETSOLKB (see Section 9.2.4), we have that $(\mathcal{K} \setminus \mathcal{D}_{\max}) \cup U_{P \cup P'}$ is returned by the algorithm.

Propositions (1) and (2) follow from the specification of the GETMODE function which is called with arguments $\langle \mathbf{D}_{\checkmark}, p_{\mathbf{D}}() \rangle$. Proposition (3) is true since GETSOLKB can never be reached without $p_{\mathbf{D}}(\mathcal{D}_{\max}) \geq 1 - \sigma$ being fulfilled. $\mathbf{D}_{\checkmark} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ is true due to Corollary 12.8, Remark 9.5 and the fact that $\mathbf{D}_{\checkmark}$ is obtained as an output of DYNAMICHS. Hence, Proposition (4) holds. Proposition (5) is implied by Remark 9.5 and by the specification of the GETFORMULAPROBS function which computes $p_{\mathcal{K}}()$ from $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}()$ as per Formulas 4.2 and 4.7 in line 5. Finally, Proposition (6) is a consequence of the definition of the GETPROBDIST function which accounts for the computation of $p_{\mathbf{D}}()$ from $p_{\mathcal{K}}()$, the input DPI, $\mathbf{D}_{\checkmark}$ and the chronological sequence of all queries and associated answers QA so far. Therefore, Proposition 9.1 is true for $mode = dynamic$.

Next, we show that the solution to Interactive Static KB Debugging is found for $\sigma = 0$ in case $mode = static$:

(s1) $\mathbf{D}_{\checkmark} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ holds for the output of STATICHS in each iteration by Proposition 11.1. Therefore, $\mathbf{D}_{\checkmark}$ comprises only minimal diagnoses w.r.t. the input DPI that comply with all specified test cases in P' and N'.

(s2) By $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}() : \widetilde{\mathcal{K}} \cup \overline{\mathcal{K}} \to (0, 1]$ we derive by Formula 4.2 that each formula in $\mathcal{K}$ must have a probability greater than zero. Further, by Formula 4.7, no formula in $\mathcal{K}$ can have a probability greater than or equal to 0.5 (i.e. in particular a probability of 1 is not possible for a formula). Hence, we have that $p_{\mathcal{K}} : \mathcal{K} \to (0, 0.5)$ for the measure $p_{\mathcal{K}}()$ computed by GETFORMULAPROBS in line 5 in Algorithm 5. Thence, by the definition of $p_{nodes}()$ in STATICHS based on $p() := p_{\mathcal{K}}()$ (cf. Definition 4.9 on page 73) due to the fact that $p_{\mathcal{K}}()$ is given as an input argument to STATICHS in line 8, we have that no diagnosis can have an (a-priori) probability of zero. Since the function GETPROBDIST might only perform some multiplications of a diagnosis probability by $\frac{1}{2}$, also the a-posteriori probability of each diagnosis must be greater than zero.

(s3) Hence, due to $\sigma = 0$, it must be necessarily be true that $|\mathbf{D}_{\checkmark}| = 1$ before the algorithm terminates.

(s4) By Problem Definition 6.2 and the specification of the GETSOLKB function, the output solution KB must be the solution to Interactive Static KB Debugging.

That a solution found for $\sigma > 0$ in case $mode = static$ might be an approximate solution to Interactive Static KB Debugging is a direct consequence of the definition of approximate solution given in Remark 9.2.

Finally, the proof that the solution to Interactive Dynamic KB Debugging is found for $\sigma = 0$ in case $mode = dynamic$ is analogue to the one for $mode = static$, just

(d1) $\mathbf{D}_{\checkmark} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ holds for the output of DYNAMICHS in each iteration by Corollary 12.8. Therefore, $\mathbf{D}_{\checkmark}$ comprises only minimal diagnoses w.r.t. the current DPI.

(d2) By (s2), (s3), Problem Definition 6.1 and the specification of the GETSOLKB function, the output solution KB must be the solution to Interactive Dynamic KB Debugging.

That a solution found for $\sigma > 0$ in case $mode = dynamic$ might be an approximate solution to Interactive Dynamic KB Debugging is a direct consequence of the definition of approximate solution given in Remark 9.2.

This completes the proof of Proposition 9.1. □

Next, we examine the complexity of Algorithm 5.[26] To this end, we denote in the following by *expensive operation* a call of a (usually) expensive function such as one that internally consults a logical

---

[26]Considerations in the rest of this section rely on the assumption that P $\neq$ NP (cf. http://bit.ly/1lIuNcP).

reasoner or another operation such as addition or multiplication that is the most time consuming algorithmic action within a certain part of an algorithm. We analyze Algorithm 5 in terms of the number num of expensive operations that are required during its execution in the worst case. The worst case time required by Algorithm 5 is then the multiplication of the maximal worst case time consumption of any expensive operation throughout the algorithm by num.

The next propositions assume $|\mathcal{K}|$ as an upper bound of $|P'| + |N'|$. This is plausible in the light of evaluations performed in e.g. [SFFR12, RSFF13] which substantiate that usually the size of the faulty KB exceeds the number of queries that are necessary to solve the interactive debugging problem by several orders of magnitude.

We first investigate the complexity of the function GETPROBDIST which is called once in each iteration of Algorithm 5:

Proposition 9.2. *Let $|\mathcal{K}|$ be an upper bound of $|P'| + |N'|$. Then, the function GETPROBDIST in Algorithm 5 requires a number of expensive operations that is linear in $|\mathcal{K}|$.*

Proof. The time complexity of GETPROBDIST can be assessed by adding the complexities of (i) GET-PRIODIAGPROBS, (ii) the for-loop between line 30 and 41, (iii) the summation in line 42 and (iv) the for-loop in lines 43 and 44. Time complexity of (i) is in $O(n_{\max} |\mathcal{K}|)$ since $|\mathbf{D}_{\checkmark}| \leq n_{\max}$ where $n_{\max}$ is a predefined constant and $|\mathcal{K}| - 1$ multiplications must be conducted per diagnosis in $\mathbf{D}_{\checkmark}$. (ii) requires $|QA| |\mathbf{D}_{\checkmark}| \leq (|P'| + |N'|) n_{\max} \leq |\mathcal{K}| n_{\max}$ many calls to functions GETENTAILMENTS and ISKBVALID, respectively, that internally call a logic reasoner. Time requirements of (iii) amount to $O(|\mathbf{D}_{\checkmark}|) = O(n_{\max})$ summations. Finally, (iv) involves $O(n_{\max})$ multiplications.

Thus, we obtain an overall time complexity of $O(n_{\max} |\mathcal{K}| + n_{\max} |\mathcal{K}| + n_{\max} + n_{\max}) = O(|\mathcal{K}|)$ for GETPROBDIST. $\square$

The next proposition is based on this result and witnesses that Algorithm 5 requires only a quadratic number of expensive operations in the size of the KB $\mathcal{K}$.

Proposition 9.3. *Let $|\mathcal{K}|$ be an upper bound of $|P'| + |N'|$ and let the function $qsm()$ given as input to Algorithm 5 be such that the time complexity of UPDATEQDATA is in $O(|\mathcal{K}|)$. Minus the time consumed by diagnosis computation (by STATICHS in case of $mode = static$ or by DYNAMICHS otherwise), the time complexity in terms of number of required expensive operations of Algorithm 5 is quadratic in $|\mathcal{K}|$.*

Proof. Variable instatiation (lines 1-4) and variable update (lines 18-26) is in $O(1)$ where some query selection measure $qsm()$ is supposed to be used, for which the time complexity of UPDATEQDATA is in $O(|\mathcal{K}|)$ (this holds for all query selection measures described in Section 9.3). GETFORMULAPROBS called in line 5 runs in $O(|\mathcal{K}| |ax_{\max}|)$ as Formula 4.2 is applied once to each formula in $\mathcal{K}$ for each of which at most $|ax_{\max}|$ multiplications are performed where $|ax_{\max}|$ is the maximum size of a formula in $\mathcal{K}$ in terms of included syntactical elements (multiple occurrences of one and the same symbol are counted multiply). As shown by Proposition 9.2, the complexity of GETPROBDIST called in line 11 is in $O(|\mathcal{K}|)$. Execution of GETMODE needs one iteration over all diagnoses in $\mathbf{D}_{\checkmark}$ in order to determine the one with maximum probability, i.e. it runs in $O(|n_{\max}|) = O(1)$ time since $n_{\max}$ is a constant. Next, GETSOLKB which computes a solution KB from a given diagnosis $\mathcal{D}$ works in $O(|\mathcal{D}| + |P| + |P'|) \subseteq O(|\mathcal{K}|)$ since $|\mathcal{D}|$ elements need to be deleted from a set of cardinality $\mathcal{K}$ which can be accomplished in constant time per element (e.g., using a hashtable) and additionally at most $|P| + |P'|$ set union operations are required, namely the union of $(\mathcal{K} \setminus \mathcal{D})$ with $U_{P \cup P'}$ where the latter needs $|P| + |P'| - 1$ set union operations. As $|P|$ is a constant c, $O(|\mathcal{D}| + |P| + |P'|) \subseteq O(2c |\mathcal{K}|) \subseteq O(|\mathcal{K}|)$. In Section 8.5, we have already underlined that GETPOOLOFQUERIES is a fixed parameter tractable problem, i.e. it requires

$$O\left(\left(n_{\max} + \left|Q_{\min}^{(\max)}\right| \log \frac{|Q^{(\max)}|}{\left|Q_{\min}^{(\max)}\right|}\right) 2^{n_{\max}}\right) = O(1)$$

calls to a reasoner in the worst case (cf. Proposition 8.9). Similarly, SELECTQUERY involves $O(2^{n_{\max}})$ comparisons $qsm(Q_i) < qsm(Q_j)$ for $Q_i, Q_j \in \mathbf{QP}$ since the cardinality of the computed query pool is in $O(2^{n_{\max}})$. The latter holds due to Proposition 8.10 which substantiates that the calculated query pool includes at most one query Q for which $\mathbf{D}^+(Q) = Y$ for each $Y \subset \mathbf{D}_{\checkmark}$. And, an upper bound for the cardinality of $\mathbf{D}_{\checkmark}$ is the constant $n_{\max}$. Therefore, the runtime of SELECTQUERY is in $O(1)$, too.

Since adding up a number of time complexities each of which is at most in $O(|\mathcal{K}|)$, we can conclude that the runtime of one iteration of Algorithm 5 minus the time needed for diagnosis computation is also in $O(|\mathcal{K}|)$, i.e. linear in $|\mathcal{K}|$ in terms of number of expensive operations needed. As there might be a maximum of $|\mathcal{K}|$ iterations by the premise that $|P'| + |N'| \leq |\mathcal{K}|$, we obtain an overall time complexity – minus the complexity of diagnoses computation – of $O(|\mathcal{K}|^2)$ for Algorithm 5. $\qquad\square$

That is, Algorithm 5 requires only a quadratic number of expensive operations "outside" of the methods STATICHS or DYNAMICHS, respectively, that account for diagnosis computation. That the substantial complexity of Algorithm 5 lies in the computation of diagnoses, is confirmed by the following results.

The first result is based on the fact that determining minimal diagnoses w.r.t. a DPI is an MBD problem (cf. page 8) which in turn can be regarded as an abduction problem as defined in [BATJ91]. More precisely, the problem of detecting minimal diagnoses w.r.t. a DPI is a *monotonic* abduction problem [BATJ91]. Hence, the following proposition holds [BATJ91, Theorem 4.3]:

Proposition 9.4. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI over $\mathcal{L}$ and let ISKBVALID (see Algorithm 1) be a function computable for $\mathcal{L}$ in polynomial time w.r.t. the size of $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ (cf. the description of the function e in [BATJ91, Section 3.3]). Then, given a set $\mathbf{D}$ of minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ such that $\emptyset \subset \mathbf{D}$, it is NP-complete to determine whether there is a minimal diagnosis $\mathcal{D}$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ such that $\mathcal{D} \notin \mathbf{D}$.*

Remark 9.11 The function ISKBVALID in the case of KB debugging is analogue to the function e used in [BATJ91]. Given the overall data D_{all} that must be explained by a solution to an abduction problem, the function e computes for a subset H of H_{all}, the set of all individual hypotheses, the set $e(H) = D$ where $D \subseteq D_{all}$ is the data explained by H. H is an explanation of the abduction problem iff it is set-minimal and $e(H) = D_{all}$ [BATJ91].

In the case of our KB debugging system, given a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, D_{all} corresponds to the set of all requirements in R and all test cases in N violated by $\mathcal{K} \cup \mathcal{B} \cup U_P$. H_{all} corresponds to $\mathcal{K}$. So, e corresponds to ISKBVALID since ISKBVALID is given some $\mathcal{K} \setminus \mathcal{D}$ and $\langle \cdot, \mathcal{B}, P, N \rangle_R$ (where $\mathcal{D}$ corresponds to some $H \subseteq H_{all}$) and checks whether $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup U_P$ does not violate any requirement or test case, i.e. whether $e(H) = D_{all}$. Notice that ISKBVALID can easily be slightly modified to return the subset of D_{all} that is explained by H, i.e. the subset of the initially violated requirements and test cases that are resolved by deletion of $\mathcal{D}$ from $\mathcal{K} \cup \mathcal{B} \cup U_P$. To this end, the early termination in case of detected invalidity must simply be omitted. $\qquad\square$

Remark 9.12 An abduction problem is monotonic [BATJ91] iff for all $H, H' \subseteq H_{all}$ it holds that $H \subseteq H' \rightarrow e(H) \subseteq e(H')$. That parsimonious KB debugging (or the problems given by Problem Definitions 3.2, 6.2, 6.1, 9.2 and 9.1) seen as an abduction problem is indeed monotonic is a simple consequence of the monotonicity of the logic $\mathcal{L}$ over which a DPI must be defined (as per the postulations of Chapter 2). For, if $(\mathcal{K} \setminus \mathcal{D}') \cup \mathcal{B} \cup U_P \models x$, then also $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup U_P \models x$ for $\mathcal{D} \subseteq \mathcal{D}'$. Modeling requirements $r \in R$ as unwanted entailments of the correct KB (see Remark 3.2), we immediately see that $\mathcal{D}$ cannot resolve more unwanted entailments $x \in R \cup N$ than $\mathcal{D}'$. Thence, parsimonious KB debugging is a monotonic abduction problem. $\qquad\square$

Unfortunately, ISKBVALID is not tractable (i.e. computable in polynomial time) for many logics $\mathcal{L}$. In particular, it is already in $\Delta_2^P = P^{NP}$ for PL (cf. the polynomial hierarchy defined by [MS72]). This

holds since propositional satisfiability checking is NP-complete [Coo71, Kar72] and since ISKBVALID, in order to to check the validity (see Definition 3.3) of a set of PL formulas X w.r.t. some PL DPI $\langle \cdot, \mathcal{B}, P \cup P', N \cup N' \rangle_R$, requires a polynomial number of calls to a propositional satisfiability checker Alg_SAT. For, by the definition of ISKBVALID (see Algorithm 1), one call of Alg_SAT is required for testing whether $X \cup \mathcal{B} \cup U_{P \cup P'}$ is consistent and a maximum of $|N| + |N'|$ further calls are needed to verify whether $X \cup \mathcal{B} \cup U_{P \cup P'} \cup \{\neg n\}$ is consistent for all $n \in N \cup N'$, i.e. whether $X \cup \mathcal{B} \cup U_{P \cup P'} \not\models n$ for all $n \in N \cup N'$ (note that $\neg n$ refers to the formula $\neg ax_1 \vee \cdots \vee \neg ax_k$ if $n := \{ax_1, \ldots, ax_k\}$, cf. page 27). Since we assume $|P'| + |N'| \leq |\mathcal{K}|$ and since $|N|$ is a constant throughout the execution of Algorithm 5, we have that the number $|N| + |N'| + 1 \leq |N| + |\mathcal{K}| + 1$ of calls to Alg_SAT performed by ISKBVALID is bounded by a polynomial in $|\mathcal{K}|$.

As a conclusion of this discussion and Proposition 9.4, we have:

Corollary 9.1. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a PL DPI given as an input to Algorithm 5. Then, each call of* STATICHS *or* DYNAMICHS *within Algorithm 5 must solve (at least) an NP-complete problem by means of an oracle that requires a polynomial number of calls to another NP-complete oracle.*

Proof. Both STATICHS and DYNAMICHS must return a set of at least $n_\mathrm{min} \geq 2$ minimal diagnoses each time they are called (given that n_min minimal diagnoses exist w.r.t. the given DPI) due to the specification of input parameter n_min in Algorithm 5 and the calls of STATICHS and DYNAMICHS in lines 8 and 10, respectively. For the first call, this implies that at least two minimal diagnoses must be found. Hence, Proposition 9.4 applies to the complexity of finding the second minimal diagnosis during the execution of the first call of both STATICHS and DYNAMICHS, just that ISKBVALID does not terminate in polynomial time, but uses a polynomial number of calls to an NP-complete oracle (the propositional satisfiability checker).

In each subsequent call of any of the two methods STATICHS and DYNAMICHS, the existing set of leading diagnoses will contain at least one minimal diagnosis w.r.t. the current DPI (since each query leaves valid at least one leading diagnosis, cf. Definition 7.1), and at least one further minimal diagnosis w.r.t. this DPI must be extracted (cf. bullet (aii) in the characterization of the outputs of STATICHS and DYNAMICHS on page 128 ff.). Thus, Proposition 9.4 holds for the computation of the first diagnosis in any subsequent call of any of the two functions, just that ISKBVALID does not terminate in polynomial time, but uses a polynomial number of calls to an NP-complete oracle (the propositional satisfiability checker). $\square$

The general complexity of ISKBVALID is even worse if DPIs over more expressive logics such as OWL 2 are considered for which one single call of a reasoner invoked by ISKBVALID is already 2-NEXPTIME-complete [GHM$^+$08, Kaz08].

However, in spite of these discouraging *theoretical* complexity results, debugging techniques similar to the ones discussed in this work have proven to perform reasonably *in practice* for many real-world KB debugging problems over DL and OWL languages, respectively [SFFR12, RSFF13, SFRF14c] which are more expressive than PL. For instance, we have shown in [SFFR12] that faulty real-world OWL KBs with sizes of up to over 33000 formulas are efficiently interactively debuggable with similar methods as those presented in this work (reaction time of the system, i.e. time between two successive queries: only 1 minute; average query length: not more than 4 formulas; overall number of queries: at most 14). Moreover, we have demonstrated in [RSFF13] that a pair of real-world OWL KBs (the first including over 11000 formulas, the second almost 5000) that has been automatically integrated by diverse ontology matching systems resulting in a faulty aligned KB (see Chapter 32 for details; we also list some matching systems there) can be debugged with absolutely reasonable time and query answering effort for the interacting user. In concrete terms, the RIO debugging strategy proposed in [RSFF13] (which can also be plugged in as a query selection measure into the system described in this work, see Section 9.3) involved an average reaction time of no more than 13 seconds and required an average number of queries to be answered by the user of no more than nine.

Chapter 10

Summary

In this part we dealt with how the process of KB debugging can be designed so as to enable a (group of) user(s) to interact with the debugging software in order to achieve high quality solutions. We defined the problem of *interactive static KB debugging* as well as the problem of *interactive dynamic KB debugging* which "naturally" arise from the fact that the DPI in interactive KB debugging is always renewed after a new test case has been specified (a new query has been answered). The former problem searches for a solution KB *w.r.t. the original DPI given as input* such that this solution KB satisfies all test cases added during the debugging session and there is no other such solution KB. The latter problem searches for a solution KB *w.r.t. the current DPI* (i.e. the original DPI including all new test cases added throughout the debugging session so far) such that there is no other solution KB w.r.t. the current DPI.

We specified the pivotal notion of a *query* which constitutes the "interface" between the debugging system and the interacting user. Queries are sets of logical formulas satisfying the *search space restriction* as well as the *solution preservation* property. That is, incorporation of any answer to a particular query into the debugging process leads to a reduction of the search space for solutions on the one hand, but guarantees the existence of at least one remaining solution on the other hand. Queries are generated from a set of *leading diagnoses* that act as a representative of all (minimal) diagnoses. We established that, for any set of at least two leading diagnoses, a query exists. The unique *q-partition* of a query constitutes the relationship between a query and the set of leading diagnoses and can be used to decide for a set of logical formulas whether this set is or is not a query. Furthermore, the q-partition can be used to estimate the impact of a query answer on the (distribution of the) set of solutions and thence can be exploited to assess the (expected) quality of different queries which in turn can help to filter out a suitable query among a pool of possible queries.

It was also presented how a pool of queries can be generated for a given set of leading diagnoses and a DPI. We showed how to minimize these queries in terms of the included number of logical formulas the aim of which is to strain the user(s) as little as possible when it comes to answering them. Moreover, we pointed out that query generation is a fixed parameter tractable problem due to the fact that the (maximum) number of leading diagnoses can be predefined and therefore constitutes a constant value (which is not growing as the diagnosis problem instance grows). We featured an in-depth discussion of the properties of the query generation algorithm, in the course of which we detected several drawbacks. The gave a hint to potential solutions that we will address in our future work. Additionally, we formally proved the correctness of the query generation method and derived complexity results. All of this was concretized by means of several illustrating examples.

Finally, we explicated the central algorithm of this work which implements an interactive KB debugging system. First, an overview of the workflow of interactive KB debugging was given, followed by a more comprehensive detailed specification of the algorithm. Some query selection measures (all of

which are later covered in more depth in Parts IV and V) were discussed and optimization versions of the problems of interactive dynamic and static KB debugging were defined where the goal is to obtain the solution to these problems by asking the user a minimal number of queries. Finally, we formally proved the correctness of the interactive KB debugging algorithm and gave a discussion of its complexity.

Part III

Iterative Diagnosis Computation

In this part we introduce and discuss two methods, STATICHS and DYNAMICHS, which are called in lines 8 and 10 of Algorithm 5, respectively. The former provides a method for solving the Interactive Static KB Debugging Problem (Problem Definition 6.2) whereas the latter aims at solving the Interactive Dynamic KB Debugging Problem (Problem Definition 6.1). Both are methods for iterative diagnosis computation that are employed to compute a set of leading diagnoses in each iteration of the presented interactive KB debugging algorithm (Algorithm 5). Each time a query has been answered by the interacting user and added to the respective set of test cases of the DPI, a subset of the leading diagnoses (and usually also a set of not-yet-computed minimal diagnoses) is invalidated. An iterative diagnosis computation method is then invoked to update the leading diagnoses set taking the new information into account that is given by the recently added test case. That is, the $k \leq n_{\max}$ most probable ways of solving the Interactive Static (Dynamic) KB Debugging Problem in the light of the new evidence are extracted by STATICHS (DYNAMICHS) after the search space has been suitably pruned. In this vein, if there is only one solution left, the (exact) solution of Interactive Static (Dynamic) KB Debugging has been found.

Chapter 11 provides an in-depth description of the *static* method and proves its correctness. Chapter 12 details the *dynamic* method and demonstrates its correctness. The practically oriented reader or the one that is willing to believe that the presented iterative diagnosis computation techniques in fact work as claimed might skip Sections 11.4 as well as 12.4 in this part.[27]

---

[27]Parts of Part III already appeared in [Rod15]. However, [Rod15] includes a significantly less detailed presentation of the algorithms and does not give any proofs of correctness.

Chapter 11

STATICHS: A Static Iterative Diagnosis Computation Algorithm

As the name already suggests, STATICHS (Algorithm 7) is a procedure that solves the problem of *Interactive Static KB Debugging* defined by Problem Definition 6.2 if used for leading diagnosis computation in Algorithm 5. STATICHS is sound, complete and optimal w.r.t. the set of solutions of the *Interactive Static KB Debugging* problem (this will be proven in Section 11.4). Optimality refers to the best-first computation of minimal diagnoses regarding a given probability measure.

11.1 Overview and Intuition

The STATICHS algorithm is strongly related to the non-interactive hitting set algorithm HS (see Algorithm 2) in that, at any stage during the execution of Algorithm 5, the hitting set tree produced by STATICHS corresponds to some part of the complete (non-interactive) wpHS-tree built-up by Algorithm 2. This is achieved by the strategy to *use new test cases only for the invalidation of diagnoses, and not for the computation of conflict sets (and thus diagnoses)*. That is, all minimal conflict sets are computed w.r.t. the input DPI. Thereby, the introduction of new diagnoses, i.e. ones that are not minimal diagnoses w.r.t. the input DPI, through addition of new test cases to the DPI is prohibited (cf. Proposition 4.6).

So, what STATICHS as a subroutine of Algorithm 5 does is gradually building up the standard (non-interactive) wpHS-tree in multiple phases. During each phase some new (not-yet-computed) minimal diagnoses w.r.t. the *input DPI* are computed, in the order of their probability, most probable ones first. Before such a newly detected minimal diagnosis is added to the set of leading diagnoses ($\mathbf{D}_{calc} \cup \mathbf{D}_{\checkmark}$), a test is performed that verifies that this new diagnosis is consistent with all test cases added to the input DPI so far. In this vein, all answered queries so far not only serve to eliminate a subset of the set of leading diagnoses at the time when the respective query is answered, but also to eliminate incompatible minimal diagnoses w.r.t. the input DPI that are found at some later point in time. However, in order to be eliminated due to a specified test case, a minimal diagnosis must first be computed. That is, no partial diagnoses can be eliminated due to newly specified test cases.

Between each two phases of tree construction, a query computed on the basis of the current set of leading diagnoses is asked to the user (this is accomplished directly in Algorithm 5). After incorporating the user's answer, some leading diagnoses are eliminated (this is granted by the definition of a query, see Definition 7.1). Moreover, the "state" of the tree is maintained during the execution of Algorithm 5 until STATICHS is again called in order to calculate further leading diagnoses. The state of the current partial

wpHS-tree is stored by variables

- $\mathbf{D}_{calc} \cup \mathbf{D}_{\checkmark}$ – computed minimal diagnoses w.r.t. the input DPI consistent with all test cases specified so far,

- $\mathbf{Q}$ – the list of open, non-labeled nodes,

- $\mathbf{C}_{calc}$ – minimal conflict sets w.r.t. the input DPI computed so far and

- $\mathbf{D}_{\times}$ – computed minimal diagnoses w.r.t. the input DPI not consistent with all test cases specified so far.

Each time a tree construction phase, i.e. the computation of new leading diagnoses, is finished, a new diagnosis probability distribution is obtained by the diagnosis probability update as per Bayes' Theorem described in Section 9.2. Once this distribution involves one highly probable diagnosis (the probability of which exceeds a predefined threshold $1 - \sigma$) and else just highly improbable ones, the algorithm terminates. The output is a solution KB w.r.t. the *input DPI* built from this highly probable minimal diagnosis.

Remark 11.1   In case σ has a predefined value of zero, the output is the (exact) solution to the problem of *Interactive Static KB Debugging* for the input DPI. In a scenario where some fault tolerance $\sigma > 0$ is given, the solution KB returned by Algorithm 5 is an approximation of the (exact) solution to *Interactive Static KB Debugging* for the input DPI where a better approximation can be expected for smaller values of σ (cf. Remark 9.2). "Better" in this context refers to the satisfaction of desired semantic properties of the KB returned by Algorithm 5, i.e. desired entailments and desired non-entailments of the KB. The intuition is that the specification of additional test cases T guarantees the output of a KB complying with these test cases, whereas accepting one – albeit highly probable – of multiple solution KBs without having incorporated T leaves open the possibility for this KB to not fulfill T.

However, answering queries is effort for an interacting user. Therefore, the approach that involves the "early" termination of the algorithm after a solution KB has a sufficiently high probability (lower than 1) constitutes a trade-off between exactness of the output and the effort of the user and overall execution time of the interactive KB debugging algorithm, respectively.   □

Constant "Convergence" towards the Solution.   As said, each added test case is an answered *query* and thus eliminates at least one minimal diagnosis w.r.t. the input DPI. And, only minimal diagnoses w.r.t. the input DPI are computed by STATICHS. Hence, by the fact that a solution to Interactive Static KB Debugging can only be constructed from a minimal diagnosis w.r.t. the input DPI, it is guaranteed that the number of solutions to Interactive Static KB Debugging is strictly monotonically decreasing throughout the execution of Algorithm 5. That is, the initial number of (all) minimal diagnoses (w.r.t. the input DPI) is "static" which means that no "new" minimal diagnoses can be introduced when the input DPI is extended by new test cases.

As a consequence of this, it is reasonable to employ STATICHS in a situation where the (complete) wpHS-tree produced by the standard (non-interactive) algorithm HS is believed to be as compact as to fit into the available system memory. In this case, STATICHS is also guaranteed to not exceed the available memory, even if an exact solution ($\sigma = 0$) is intended.

Unfortunately, however, it will be generally the case that a complete enumeration of all minimal diagnoses is intractable, especially due to an overwhelming space complexity. In such a case, Algorithm 5 using STATICHS will definitely run out of memory (given that STATICHS is called sufficiently often). The reason is that the space consumption of STATICHS will sooner or later definitely reach the huge extent of the wpHS-tree produced by HS. Nevertheless, STATICHS might be used to (possibly) find some (approximate) solution. This might work in a scenario where the given probabilistic information in terms

of $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}()$ provided as an input to Algorithm 5 is "reasonable" in that the desired diagnosis is assigned a rather high probability and is thus figured out early, before the available memory is exhausted.

A possible modification of the stop criterion in STATICHS in a way that new leading diagnoses are not computed until a desired number of such is detected or a timeout is reached, but rather until a predefined maximum space is consumed, would not mitigate space complexity issues very much. An explanation for this is that stopping STATICHS on account of no more available memory implies that no further call of STATICHS will be able to execute. That is because, as mentioned before, an added test case can only invalidate already computed diagnoses, no other branches in the wpHS-tree, and each invalidated minimal diagnosis cannot be discarded, but must be stored (in $\mathbf{D}_\times$) to avoid the usage of leading diagnoses that are non-minimal w.r.t. the input DPI (cf. lines 21-23 in Algorithm 7).

Poor Search Tree Pruning. As we explained before, the preservation of a constantly shrinking set of minimal diagnoses comes at the cost of being able to exploit new test cases only partially, i.e. only for the invalidation of already computed minimal diagnoses w.r.t. the input DPI and not for the computation of minimal conflict sets and thus minimal diagnoses. The incorporation of test cases into the DPI that is used to determine minimal conflict sets (line 30 in Algorithm 7) could, on the one hand, lead to new minimal conflict sets that are no minimal conflict sets w.r.t. the input DPI. As a consequence of this, minimal diagnoses might be determined by the algorithm which are no minimal diagnoses w.r.t. the input DPI, but w.r.t. the current DPI. Hence, the soundness of STATICHS w.r.t. the set of solutions of the Interactive Static KB Debugging problem would be violated. Furthermore, such conflict sets could lead to the missing of some minimal diagnoses w.r.t. the input DPI, a violation of the completeness of STATICHS w.r.t. the set of solutions of the Interactive Static KB Debugging problem.

On the other hand, the exploitation of new test cases for conflict set generation might give rise to the possibility of pre-pruning of *any* tree branches, not just branches that already correspond to diagnoses w.r.t. the input DPI. Such a "dynamic" strategy which exploits the new information given by a test case not just partially, but for the invalidation *and computation* of diagnoses and conflict sets, will be implemented be DYNAMICHS which we will detail in Chapter 12.

Put another way, in STATICHS only the standard pruning rules for the construction of a wpHS-tree are applicable, namely the deletion of duplicate nodes and the elimination of non-minimal diagnoses (cf. Definition 4.10). Newly defined test cases only facilitate the deletion of tree branches from the leading diagnoses set $\mathbf{D}_{calc} \cup \mathbf{D}_\checkmark$, but not from memory (as invalidated minimal diagnoses must be stored in $\mathbf{D}_\times$, as pointed out before).

To summarize, STATICHS on the one hand makes sure to only consider relevant solutions of the problem of Interactive Static KB Debugging, but on the other hand suffers from this conservative strategy in that tree pruning cannot be designed very effectively. So, on the positive side, uncontrolled growth of the produced wpHS-tree can be avoided, but, on the negative side, consultation of an interacting user cannot be taken advantage of in terms of reduction of the space complexity of STATICHS compared to the construction of a wpHS-tree by a non-interactive procedure like Algorithm 2.

11.2 Algorithm Walkthrough

Input Parameters. When STATICHS (Algorithm 7) is called for the first time in Algorithm 5, the inputs $\mathbf{C}_{calc}$, $\mathbf{D}_\checkmark$, $\mathbf{D}_\times$, P' and N' correspond to the empty set and $\mathbf{Q} = [\emptyset]$ (cf. lines 1-4 and 8 in Algorithm 5). Further on, $\mathbf{D}_{calc}$ is defined to be the empty set at the beginning of each execution of STATICHS. That is, STATICHS starts the construction of the wpHS-tree from an initial tree consisting of a single unlabeled root node $\emptyset$ ($\in \mathbf{Q}$). And, all collections that are later returned by STATICHS, except for $\mathbf{Q}$, are initially empty. Further input arguments are the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ provided as an input to Algorithm 5, the sets of positively (P') and negatively (N') answered queries since the start of Algorithm 5, the leading

diagnosis computation parameters $n_{\min}, n_{\max}, t$ (see the description in Chapter 7 on page 95) and the probability measure $p() := p_{\mathcal{K}}()$ that assigns a probability in the interval $(0, 0.5)$ to each formula in $\mathcal{K}$ (cf. line 5 in Algorithm 5).

The Main Loop. During the repeat-loop, in each iteration the first node node in $\mathbf{Q}$ is processed (GETFIRST, line 5). That is, node is deleted from $\mathbf{Q}$ (DELETEFIRST, line 6) and the SLABEL function is called given node (i.a.) as a parameter. Notice that elements are added to $\mathbf{Q}$ (line 17) in a way that a sorting of $\mathbf{Q}$ in descending order according to $p_{nodes}()$ (cf. Definition 4.9) is maintained throughout the execution of STATICHS.

Computation of a Node Label. The SLABEL function processes node as follows. First, the *non-minimality criterion* (lines 21-23) is checked. That is, among all nodes in $\mathbf{D}_{(\times,\checkmark,calc)} = \mathcal{D}_{\times} \cup \mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc}$ one is searched which is a subset of node. If such a node nd is found, then node must be a non-minimal diagnosis (nd $\subset$ node) or a duplicate diagnosis (nd $=$ node) w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ since all sets $\mathcal{D}_{\times}, \mathbf{D}_{\checkmark}$ and $\mathbf{D}_{calc}$ contain only minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. In this case, the branch in the wpHS-tree corresponding to node can be dismissed which is taken account of by returning the label *closed* for node.

In case the non-minimality criterion is not satisfied, the *duplicate criterion* (lines 24-26) is checked next. Here, $\mathbf{Q}$ is browsed for a node that is equal to node. If such a one is found, node can be discarded because it suffices to consider only one tree branch among multiple tree branches in the wpHS-tree featuring one and the same set of edge labels. Hence, *closed* is returned as a label for node. Altogether, this means that only the last processed exemplar of a node corresponding to one and the same set of edge labels is labeled, all others are discarded.

If the duplicate criterion is not met, the *reuse criterion* (lines 27-29) is checked next. That is, $\mathbf{C}_{calc}$ is browsed for a set $\mathcal{C}$ ($\mathbf{C}_{calc}$ comprises only minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$) such that $\mathcal{C}$ and node are disjoint sets. If such a $\mathcal{C}$ is detected, then $\mathcal{C}$ can be used to label node since the set of edge labels along the path in the wpHS-tree leading from the root node to node does not hit $\mathcal{C}$. In this case, the label $\mathcal{C}$ is returned for node by SLABEL.

Given that the reuse criterion fails, QX is called given the DPI $\langle \mathcal{K} \setminus \text{node}, \mathcal{B}, P, N \rangle_R$ as an argument (line 30). If the output L is equal to 'no conflict', then we know by Proposition 4.9 that node is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, wherefore the label *valid* is returned for node. Otherwise, the output L must be a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that has an empty set-intersection with node. Since the reuse criterion failed, i.e. there is no set in $\mathbf{C}_{calc}$ that does not intersect with node, L must be a fresh minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ in the sense that $L \notin \mathbf{C}_{calc}$ must hold. Therefore the label L is first added to $\mathbf{C}_{calc}$ and then returned by SLABEL as a label for node.

Processing of a Node Label. Back in the main procedure, $\mathbf{C}_{calc}$ is updated (line 8) and then the label L returned by the SLABEL function is processed as follows. If $L = valid$, then it is a fact that node is a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, but it is not certain that node also meets all positive test cases P' and all negative test cases N' that have been specified and added to $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ so far. Thus, according to Proposition 7.3, the validity of the KB $\mathcal{K} \setminus \text{node}$ w.r.t. $\langle \cdot, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ must still be checked (line 10). If successful, node is added to the set $\mathbf{D}_{calc}$ of calculated minimal diagnoses w.r.t. the input DPI that comply with all answered queries so far. Otherwise, node is added to the set $\mathbf{D}_{\times}$ of minimal diagnoses w.r.t. the input DPI that have been invalidated by some answered query.

Roughly, the minimality of diagnoses added to $\mathbf{D}_{calc}$ is assured by the pruning rule (lines 21-23) which eliminates non-minimal nodes and the fact that $p_{nodes}()$ sorts a node nd' corresponding to a superset of some node nd behind nd in $\mathbf{Q}$.

If, on the other hand, $L = closed$ is the label returned by SLABEL, then node must simply be removed from $\mathbf{Q}$ which has already been executed in line 6. Thence, no actions are necessary (cf. line 14).

In the third case, if a minimal conflict set L is returned by SLABEL, then L is a label for node meaning that $|L|$ successor nodes of node, namely a node node $\cup \{e\}$ for all elements $e \in L$, need to be added to $\mathbf{Q}$ in sorted order using the function $p_{nodes}()$ (INSERTSORTED, line 17).

Stop Criterion. The first criterion causing STATICHS to terminate is $\mathbf{Q} = []$ which means that the complete wpHS-tree has been constructed and no further nodes can be labeled. In this case, $\mathbf{D}_{calc} \cup \mathbf{D}_{\checkmark}$ comprises all minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that are compliant with all the specified positive and negative test cases P' and N'.

If the first criterion is not met, then the second criterion is checked. That is, a test is performed which checks whether the number of leading minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ in $\mathbf{D}_{calc} \cup \mathbf{D}_{\checkmark}$ amounts to at least $n_{\min}$ and either $|\mathbf{D}_{calc} \cup \mathbf{D}_{\checkmark}| = n_{\max}$ or more than t time has passed since the start of the execution of STATICHS. In the latter case, $n_{\min} \leq |\mathbf{D}_{calc} \cup \mathbf{D}_{\checkmark}| < n_{\max}$ holds. In the former case, $|\mathbf{D}_{calc} \cup \mathbf{D}_{\checkmark}| = n_{\max}$ is satisfied.

Processing of the Leading Diagnoses Returned by STATICHS. When a call of STATICHS in Algorithm 5 returns $\langle \mathbf{D}_{calc} \cup \mathbf{D}_{\checkmark}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times} \rangle$, the set $\mathbf{D}_{calc} \cup \mathbf{D}_{\checkmark}$ is stored in the variable $\mathbf{D}_{\checkmark}$ in Algorithm 5. Between two successive calls of STATICHS in Algorithm 5, only this set $\mathbf{D}_{\checkmark}$ as well as $\mathbf{D}_{\times}$ are modified. The list $\mathbf{Q}$ and the set $\mathbf{C}_{calc}$ remain unchanged until they are used as input parameters to the next call of STATICHS in Algorithm 5.

In case one diagnosis $\mathcal{D}_{\max}$ of the current leading diagnoses in $\mathbf{D}_{\checkmark}$ has a probability greater or equal $1 - \sigma$ as per the probability measure $p_{\mathbf{D}}()$ (see Section 9.2), the stop criterion of interactive KB debugging is met and a solution KB w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ constructed from the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ as well as from $\mathcal{D}_{\max}$ is returned to the user. Thereafter, Algorithm 5 terminates and no more calls of STATICHS take place.

Otherwise, if no leading diagnosis satisfies the stop criterion, a query Q together with its q-partition $\mathfrak{P}(Q)$ is computed, as was detailed in Chapter 8 and Section 9.2. An answer $u(Q)$ to this query is submitted by the interacting user (line 17 in Algorithm 5). Then $u(Q)$ along with $\mathfrak{P}(Q)$ is exploited to figure out the subset $\mathbf{D}_{out}$ of $\mathbf{D}_{\checkmark}$ that does not comply with $u(Q)$. This set $\mathbf{D}_{out}$ is then deleted from $\mathbf{D}_{\checkmark}$ and added to $\mathbf{D}_{\times}$. Additionally, Q is added to the positive test cases P' if $u(Q) = \mathit{true}$ and to the negative test cases N' otherwise. Subsequently, STATICHS is called again given

- the updated parameters $\mathbf{D}_{\checkmark}, \mathbf{D}_{\times}, P'$ and N' (which are modified within and outside of STATICHS during the execution of Algorithm 5),

- the unchanged parameters $\mathbf{Q}, \mathbf{C}_{calc}$ (which are modified only within STATICHS during the execution of Algorithm 5) and

- the constant parameters $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R, t, n_{\min}, n_{\max}$ and $p_{\mathcal{K}}()$ (which are not modified within or outside of STATICHS during the execution of Algorithm 5).

The execution of this next and any subsequent call to STATICHS runs in analogue way as described.

Remark 11.2 We want to emphasize that queries are computed w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ although STATICHS focuses on solutions to the problem of Interactive Static KB Debugging which involves exclusively minimal diagnoses w.r.t. the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. However, a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that satisfies all positive test cases P' as well as all negative test cases N' is also a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. And, a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that does not satisfy all positive test cases P' as well as all negative test cases N' is not a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. These two facts are guaranteed by Proposition 12.5 that will be given on page 201.

Hence, it holds that

- $\mathcal{D}$ is a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that satisfies $P' \cup \{Q\}$ as well as N' if and only if $\mathcal{D}$ is a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P' \cup \{Q\}, N \cup N' \rangle_R$ and

- $\mathcal{D}$ is a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that satisfies P' as well as $N' \cup \{Q\}$ if and only if $\mathcal{D}$ is a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \cup \{Q\} \rangle_R$.

Therefore, each query constructed during Algorithm 5 with $mode = static$ must be a query w.r.t. the current set of leading diagnoses $\mathbf{D}_{\checkmark}$ and the *current* DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ (cf. Equation 7.1, Definition 7.2 and Proposition 7.3 on pages 95-96).

   As a consequence of this, no additional test is required in order to ascertain that each diagnosis in the set $\mathbf{D}_{\checkmark}$ that is given as a parameter to the next call of STATICHS does in fact satisfy all answered queries so far. $\square$

11.3 Illustrating Examples

In this section we will give two examples of how interactive KB debugging using STATICHS (Algorithm 5 with parameter $mode = static$) works. The first one will show the similarities and differences between the usage of STATICHS (within Algorithm 5) and HS (within Algorithm 3) since it will depict the application of STATICHS on the same example DPI (see Table 15.3) that was used to show the functionality of HS in examples 4.8 and 4.9. At the same time, the first example will provide evidence that solving the problem of Interactive Static KB Debugging can be more efficient than solving the problem of Interactive Dynamic KB Debugging in terms of the number of query answers required from an interacting user. This will be discussed in more detail in Chapter 13.

   The second example is supposed to deepen the reader's understanding of the way STATICHS works. To this end, the example DPI provided by Table 4.2 will be used which constitutes a significantly harder (interactive) debugging task than the DPI investigated in the first example. This example will involve the construction of a relatively large hitting set tree and thereby give a presentiment of the space and time complexity problems caused by the poor tree pruning inherent in the STATICHS algorithm. In addition, this example will draw a reverse image of the first example in that it will stress the advantage of the decision to search for a solution of Interactive Dynamic KB Debugging rather than for a solution of Interactive Static KB Debugging (more on that in Chapter 13).

Example 11.1   In this example we assume that the author (called user throughout this example) of the (admissible) DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given by Table 15.3 applies Algorithm 5 with $mode = static$ to interactively debug $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Further, suppose the following user requirements:

   In order to guarantee a fast reaction time of the system (the time between two successive queries to the user), the user wants each query to be computed from the minimally necessary number of leading diagnoses. Thus, in each iteration exactly two leading diagnoses should be computed by STATICHS (cf. Proposition 7.5). This postulation is reflected by setting $n_{\min} = n_{\max} = 2$. Notice that the time limit t is irrelevant in this case.

   Moreover, the user desires to get just any query, i.e. they do not demand any particular properties – such as optimal information gain among a pool of queries – to be satisfied by a query. This can be ensured by choosing $q := 1$ (cf. Chapter 8) and $qsm()$ equal to any query selection measure described in Section 9.3.

   The user is new to KB debugging and has neither an idea of faults they frequently make nor access to any kind of data that would indicate their tendency to certain types of faults. Thence, $p_{\mathcal{K}}(ax) := c < 0.5$ for all $ax \in \mathcal{K}$, i.e. all formula fault probabilities are specified to be equal (to some constant c). In such a case, if a formula fault probability measure $p_{\mathcal{K}}()$ is given as an input to Algorithm 5, then line 5 in Algorithm 5 is omitted. Please notice that this aspect is not shown in Algorithm 5.

Finally, the user's intention is to get the (exact) solution to the problem of Interactive Static KB Debugging. This can be taken into account by specifying $\sigma := 0$.

The tree constructed and parameters computed and used by Algorithm 5 using STATICHS are visualized by Figure 11.1. We use the same notation as in Figures 4.2 and 4.3 which is described in Examples 4.8 and 4.9. The only new notational element here is the $\Longrightarrow$ labeled by some designator of a query. That is, $\checkmark_{(\mathcal{D}_i)} \overset{Q_j}{\Longrightarrow} \checkmark$ means that $\mathcal{D}_i$ is still a minimal diagnosis after Q_j has been answered and added to the respective set of test cases of the DPI. On the other hand, $\checkmark_{(\mathcal{D}_i)} \overset{Q_j}{\Longrightarrow} \times$ signifies that the minimal diagnosis $\mathcal{D}_i$ is invalidated through the addition of the answered query Q_j to the respective set of test cases of the DPI. Please notice that $\Longrightarrow$ does not point at a node of the wpHS-tree. Instead, the label at which $\Longrightarrow$ points is to be understood as the new label of the node originally labeled by $\checkmark_{(\mathcal{D}_i)}$ from which the (first of possibly multiple) $\Longrightarrow$ goes out. This notation should help to keep track of the evolution of node labels in the wpHS-tree without needing to overload a single node by multiple different successive labels.

In the first iteration, i.e. during the execution of the first call of STATICHS during Algorithm 5, the root node (initially the empty set) is labeled by the minimal conflict set $\langle 1, 2, 5 \rangle$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and three successor nodes, namely $\{1\}$, $\{2\}$ as well as $\{5\}$, are added to the queue of open nodes $\mathbf{Q}$. Since all formulas have been assigned an equal fault probability, STATICHS conducts a breadth-first tree construction (as displayed by the numbers ⓘ that give the order of node labeling). That is, $\mathbf{Q}$ in this case is a first-in-first-out queue. In this vein, first $[1]$ and then $[2]$ are identified as minimal diagnoses w.r.t. the given DPI. Since $\mathbf{D}_\checkmark \cup \mathbf{D}_{calc} = \emptyset \cup \{[1], [2]\}$ has a cardinality of $n_{\min} = n_{\max} = 2$, the stop criterion of STATICHS causes it to terminate and return $\langle \mathbf{D}_{calc} \cup \mathbf{D}_\checkmark, \mathbf{C}_{calc}, \mathbf{Q}, \mathbf{D}_\times \rangle = \langle \{[1], [2]\}, \{\langle 1, 2, 5 \rangle\}, [\{5\}], \emptyset \rangle$ (because $\mathbf{D}_\checkmark$ and $\mathbf{D}_\times$ are initially empty sets), as shown in the upper right column in Figure 11.1.

Then, in Algorithm 5, outside of the STATICHS procedure, the first query $Q_1 = \{E \to \neg A\}$ is computed from the leading diagnoses set $\{[1], [2]\}$. The q-partition $\mathfrak{P}(Q_1)$ associated with Q_1 is $\langle \{[1]\}, \{[2]\}, \emptyset \rangle$. The user's answer $u(Q_1)$ to Q_1 is then *false*. Thence, the set $\mathbf{D}_{out}$ is calculated from $\mathfrak{P}(Q_1)$ as $\mathbf{D}^+(Q_1) = \{[1]\}$ (due to negative answer, cf. Remark 7.4), deleted from $\mathbf{D}_\checkmark := \mathbf{D}_\checkmark \cup \mathbf{D}_{calc}$ to yield $\mathbf{D}_\checkmark = \{[2]\}$ and added to $\mathbf{D}_\times$ to yield $\mathbf{D}_\times = \{[1]\}$. The set $\mathbf{D}_\checkmark$ corresponds to the set of all already computed minimal diagnoses w.r.t. the input DPI that satisfy all queries answered so far. The set $\mathbf{D}_\times$ comprises all already computed minimal diagnoses w.r.t. the input DPI that do not satisfy all queries answered so far. These sets $\mathbf{D}_\checkmark$ and $\mathbf{D}_\times$ along with the collections $\mathbf{Q}$ and $\mathbf{C}_{calc}$ which are unmodified outside of STATICHS are used as input arguments for the second call of STATICHS. Notice that, in the figure, the resulting values of operations performed within STATICHS are given in the righthand column above the dashed line whereas values computed outside of STATICHS are given below the dashed line.

After the modifications caused by the addition of the query Q_1 to the negative test cases of $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ have been taken into account in step ④, the partial wpHS-tree built in iteration 1 is further constructed in iteration 2 resulting in the tree depicted by the middle picture in the lefthand column of Figure 11.1. Whereas the branches with edge labels $\{5, 1\}$ and $\{5, 2\}$ correspond to proper supersets of the minimal diagnoses $[1]$ and $[2]$, respectively, w.r.t. the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and are thus closed by the non-minimality criterion tested in the SLABEL function, the branch with edge labels $\{5, 7\}$ is identified as a minimal diagnosis $\mathcal{D}_3 := [5, 7]$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. However, $\mathcal{D}_3$ is not directly added to the set $\mathbf{D}_{calc}$. In fact, the validity of the KB $\mathcal{K} \setminus \mathcal{D}_3$ w.r.t. the *current* DPI $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q_1\} \rangle_R$ is tested beforehand. As this test is successful, meaning that $\mathcal{D}_3 \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q_1\} \rangle_R}$, $\mathcal{D}_3$ can be safely added to $\mathbf{D}_{calc}$ implying the set of leading diagnoses $\mathbf{D}_\checkmark \cup \mathbf{D}_{calc} = \{\mathcal{D}_2, \mathcal{D}_3\}$ with cardinality two. Due to $n_{\min} = n_{\max} = 2$, STATICHS terminates.

After the second query Q_2 has been answered negatively involving the dismissal of the leading diagnosis $\mathcal{D}_2$, STATICHS ends up with an empty queue $\mathbf{Q}$ of open nodes in iteration 3 (see the tree in the lower left column of Figure 11.1). Hence, STATICHS returns a singleton set including the leading diagnosis $\mathcal{D}_3$. Now, independently of the specified formula probabilities, $p_\mathbf{D}(\mathcal{D}_3) = 1 \geq 1 - \sigma = 1$ is satisfied since

the probability space considered by the probability measure $p_\mathbf{D}()$ focuses on the sample space $\Omega = \{\mathcal{D}_3\}$ (cf. Sections 4.6 and 9.2). Thus, the stop condition of Algorithm 5 is met wherefore the solution KB $\mathcal{K}_{sol} := (\mathcal{K} \setminus \mathcal{D}_3) \cup U_P = (\mathcal{K} \setminus \mathcal{D}_3) \cup \emptyset = \mathcal{K} \setminus \mathcal{D}_3$ is returned to the user. This solution KB $\mathcal{K}_{sol}$ is the (exact) solution to Interactive Static KB Debugging given the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ of Table 15.3 as an input because $\mathcal{D}_3$ is the only minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that conforms with all answered queries $Q_1 = false$ and $Q_2 = false$.

All in all, the execution of Algorithm 5 in this example performs

- 2 full QX calls, i.e. calls of QX that actually return a minimal conflict set (there are two minimal conflict sets labeled by C in the picture at the bottom of the lefthand column in Figure 11.1) and

- 6 validity checks, i.e. calls of QX that return 'no conflict' (one check for each of the three found minimal diagnoses; notice that QX does only perform a single KB validity check by ISKBVALID in case it returns 'no conflict', see Algorithm 1) or calls of ISKBVALID in line 10 in STATICHS (one call for each of the three found minimal diagnoses),

computes

- 3 minimal diagnoses w.r.t. the input DPI,

- 2 minimal conflict sets w.r.t. the input DPI and

- 2 queries and asks the user 2 logical formulas (1 per query)

and stores

- a maximum of 5 nodes (where node refers to the internal representation of a node in STATICHS as a set of edge labels along a path from the root node to a leaf node; there are even more nodes in the sense of tree nodes in the picture at the bottom of the lefthand column in Figure 11.1). □

Example 11.2 Let us now consider the (admissible) DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given by Table 4.2. We assume an expert (called user throughout this example) in the domain Dom modeled by $\mathcal{K}$ who wants to find a solution to Interactive Static KB Debugging for the given DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ by means of Algorithm 5 with $mode = static$. Further, we suppose the following requirements:

The user wants each query to be computed from three leading diagnoses. Thus, after each iteration of STATICHS, the set $\mathbf{D}_\checkmark \cup \mathbf{D}_{calc}$ should comprise exactly three elements. This postulation is reflected by setting $n_{\min} = n_{\max} = 3$. Notice that the time limit t is irrelevant in this case.

Moreover, as in example 11.1, we assume no demand for queries satisfying special properties which is reflected by choosing $q := 1$ (cf. Chapter 8) and $qsm()$ equal to any query selection measure described in Section 9.3.

Let there be several documentations of past debugging sessions (e.g. in terms of formula change logs) involving KBs in the domain Dom of the author $auth$ of $\mathcal{K}$ accessible to the user. Further, let the user have extracted term and logical construct probabilities $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}(ax) \in [0, 1]$ for $ax \in \mathcal{K}$ for $auth$ from this data. This function $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}} : \widetilde{\mathcal{K}} \cup \overline{\mathcal{K}} \to [0, 1]$ is then provided as an input to Algorithm 5.

Finally, the user's intention is to get the (exact) solution to the problem of Interactive Static KB Debugging. This can be taken into account by specifying $\sigma := 0$.

The tree constructed and parameters computed and used by Algorithm 5 using STATICHS are visualized by Figures 11.2 as well as 11.3. We use the same notation as in Figures 4.2, 4.3 and 11.1 which is described in Examples 4.8, 4.9 and 11.1.

After the initialization of variables, Algorithm 5 calls the function GETFORMULAPROBS in line 5 which exploits $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}()$ to calculate the function $p_{\mathcal{K}}()$ giving the fault probabilities of formulas in $\mathcal{K}$ (cf. Sections 4.6.1, 9.2 and Example 4.7). Let the resulting probabilities be as depicted by Table 11.1.

$ax \in \mathcal{K}$	1	2	3	4	5	6	8
$p_{\mathcal{K}}(ax)$	0.26	0.18	0.21	0.41	0.18	0.40	0.18

Table 11.1: (Example 11.2) Computed formula fault probabilities for the example DPI given by Table 4.2.

Then, STATICHS is called for the first time, resulting in the wpHS-tree given in the first picture in Figure 11.2. Contrary to Example 11.1, where the tree was built up in breadth-first order, in this example the formula probabilities $p() := p_{\mathcal{K}}()$ given by Table 11.1 are used to assign a probability $p_{nodes}(\mathsf{n})$ to each path n in the wpHS-tree starting from the root node (cf. Formula 4.6 and Definition 4.9). In this vein, as outlined by the numbers ⓘ indicating when a node is labeled, after the root node has been labeled by $\mathcal{C}_1 := \langle 1, 2, 5 \rangle$, the node corresponding to the outgoing edge of $\mathcal{C}_1$ labeled by the formula with the largest fault probability among all formulas in $\mathcal{C}_1$ is labeled first. That is, the node $\{1\}$ with $p_{nodes}(\{1\}) = 0.41$ (as opposed to the nodes $\{2\}$ and $\{5\}$ with 0.25 each) is labeled first. The SLABEL procedure, after checking whether $\{1\}$ is a non-minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ or a duplicate of some other node in $\mathbf{Q}$ (both checks negative), computes another minimal conflict set $\mathcal{C}_2 := \langle 2, 4, 6 \rangle$ such that $\{1\} \cap \mathcal{C}_2 = \emptyset$ ($\mathcal{C}_2$ is not hit by the node $\{1\}$) to constitute a label for node $\{1\}$. The successor nodes $\{1, 2\}$, $\{1, 4\}$ and $\{1, 6\}$ of $\{1\}$ are generated and added to the list $\mathbf{Q}$ in a way that the sorting of $\mathbf{Q}$ in descending order of $p_{nodes}()$ is maintained.

Since $\{1, 4\}$ (0.28) as well as $\{1, 6\}$ (0.27) have a larger probability (as per $p_{nodes}()$) than the nodes $\{2\}$ (0.25) and $\{5\}$ (0.25), $\mathbf{Q}$ is given by $[\{1, 4\}, \{1, 6\}, \{2\}, \{5\}, \{1, 2\}]$ when it comes to the processing of the next node. Since STATICHS always treats the first node of $\mathbf{Q}$ next, it identifies the first minimal diagnosis $\mathcal{D}_1 := [1, 4]$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ in step ③. In steps ④ and ⑧, two further minimal diagnoses $\mathcal{D}_2 := [1, 6]$ and $\mathcal{D}_3 := [5, 4]$ are detected. Altogether, the union of $\mathbf{D}_{\checkmark}$ (initially the empty set) and $\mathbf{D}_{calc}$ (comprising the three computed diagnoses) now contains $3 = n_{\min} = n_{\max}$ elements wherefore STATICHS terminates and outputs the tuple $\langle \mathbf{D}_{calc} \cup \mathbf{D}_{\checkmark}, \mathbf{C}_{calc}, \mathbf{Q}, \mathbf{D}_{\times} \rangle$ where the sets in this tuple are given under the wpHS-tree of iteration 1 in Figure 11.2.

From this set of leading diagnoses $\mathbf{D}_{\checkmark} := \mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc}$, the probability measure $p_{\mathbf{D}} : \mathbf{D}_{\checkmark} \to [0, 1]$ is computed by the function GETPROBDIST (cf. Algorithm 6 and Section 9.2). The result is $\langle p_{\mathbf{D}}(\mathcal{D}_1), p_{\mathbf{D}}(\mathcal{D}_2), p_{\mathbf{D}}(\mathcal{D}_3) \rangle = \langle 0.38, 0.37, 0.25 \rangle$. The mode $\mathcal{D}_{\max} := \mathcal{D}_1$ of this probability distribution is then computed by GETMODE. As $\sigma = 0$, $p_{\mathbf{D}}(\mathcal{D}_{\max}) = 0.38 \not\geq 1$ wherefore the stop criterion of Algorithm 5 is not satisfied.

Consequently, Algorithm 5 proceeds to generate the first query $Q_1 = \{B \sqsubseteq K\}$ (based on the current set of leading diagnoses $\mathbf{D}_{\checkmark}$) along with its associated q-partition $\mathfrak{P}(Q_1) = \langle \{\mathcal{D}_1, \mathcal{D}_2\}, \{\mathcal{D}_3\}, \emptyset \rangle$. The diagnosis $\mathcal{D}_1$ is in $\mathbf{D}^+(Q_1)$ because $\mathcal{K}_1^* = (\mathcal{K} \setminus \mathcal{D}_1) \cup \mathcal{B} \cup U_P$ (recall Formula 7.1 for a definition of $\mathcal{K}_i^*$) comprises formulas 2, 3, 5, 6, 7, 8 and 9 as well as p_1 (cf. Table 4.2) wherefore $\mathcal{K}_1^* \models \{B \sqsubseteq K\} = Q_1$ (due to the set of formulas $\{2, 3\} = \{B \sqsubseteq G, G \sqsubseteq K\}$). That $\mathcal{D}_2$ belongs to $\mathbf{D}^+(Q_1)$ as well follows analogously. On the other hand, $\mathcal{D}_3 \in \mathbf{D}^-(Q_1)$ must be true since $\mathcal{K}_3^* \cup Q_1$ includes i.a. $A \sqsubseteq B$ (formula 1) and $B \sqsubseteq K$ ($\in Q_1$) wherefore $\{A \sqsubseteq K\} = n_1$ is an entailment of $\mathcal{K}_3^*$. Thus, the negative test case n_1 is violated.

The positive user answer $u(Q_1) = true$ is incorporated in that Q_1 is appended to the set of positive test cases P yielding $P \cup \{Q_1\} = \{\{r(x, y)\}, \{B \sqsubseteq K\}\}$. Step ⑨ shows the impact of this test case addition on the set of leading diagnoses, i.e. all diagnoses in the set $\mathbf{D}_{out} = \mathbf{D}^-(Q_1) = \{\mathcal{D}_3\}$ (due to positive answer, cf. Remark 7.4) are re-labeled by $\times$ whereas all other leading diagnoses ($\mathcal{D}_1, \mathcal{D}_2$) are still labeled by $\checkmark$.

In the same fashion, further node labelings are conducted in iteration 2 until $|\mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc}| = |\{\mathcal{D}_1, \mathcal{D}_2\} \cup \{[2, 1]\}| = 3 = n_{\min} = n_{\max}$ holds again. These actions are displayed by the tree at the bottom of Figure 11.2.

Notice that, after step ⑫, two nodes corresponding to the same set are elements of the list $\mathbf{Q}$. At

step $\text{\textcircled{13}}$, the duplicate criterion checked by SLABEL comes into play. Since the node $\{1, 2\}$ (the leftmost branch in the tree) is ranked first in $\mathbf{Q}$ (we assume a first-in-first-out ordering of nodes corresponding to equal sets of edge labels in $\mathbf{Q}$), the SLABEL procedure is called given node $:= \{1, 2\}$ as an argument and detects the node $\{2, 1\}$ (the fourth leftmost branch in the tree) in $\mathbf{Q}$. Hence, node $= \{1, 2\}$ is closed as a duplicate node which finds expression in the label $\times_{(dup)}$. When $\{2, 1\}$ (which must have the same probability as $\{1, 2\}$ due to set-equality) is processed at step $\text{\textcircled{14}}$, it is discovered to be a minimal diagnosis ($\mathcal{D}_5$) w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Moreover, we want to point out that another minimal diagnosis ($\mathcal{D}_4 = [2, 4, 6]$) is found in iteration 2 before $\mathcal{D}_5$ is detected. However, $\mathcal{D}_4$ is immediately ruled out and added to $\mathbf{D}_\times$ (cf. line 13 in STATICHS) due to the fact that $\mathcal{K} \setminus \mathcal{D}_4$ is invalid w.r.t. the *current* DPI $\langle \cdot, \mathcal{B}, P \cup \{Q_1\}, N \rangle_R$ (cf. Definition 3.3). The explanation why this holds is as follows:

By Definition 3.3, $\mathcal{K} \setminus \mathcal{D}_4$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P \cup \{Q_1\}, N \rangle_R$ iff $\mathcal{K}_4^* = (\mathcal{K} \setminus \mathcal{D}_4) \cup \mathcal{B} \cup U_{(P \cup \{Q_1\})}$ (recall Formula 7.1 for a definition of $\mathcal{K}_i^*$) does not violate any $r \in R = \{\text{consistency, coherency}\}$ and does not entail any $n \in N = \{n_1, n_2\} = \{\{A \sqsubseteq K\}, \{L \sqsubseteq \exists r.F, B(x), G \sqsubseteq K\}\}$. Applying the diagnosis $\mathcal{D}_4$ to $\mathcal{K}$ yields $\mathcal{K} \setminus \mathcal{D}_4 = \{1, 3, 5, 8\}$ which includes in particular formula 1 which is equal to $A \sqsubseteq B$ (see Table 4.2). However, there is also the negative test case n_1 indicating that $A \sqsubseteq K$ must not be entailed by $\mathcal{K}_4^*$. That is, $B \sqsubseteq K \in \mathcal{K}_4^*$ (due to Q_1) and $A \sqsubseteq B \in \mathcal{K}_4^*$ which implies that $\mathcal{K}_4^* \models \{A \sqsubseteq K\} = n_1$ wherefore $\mathcal{K}_4^*$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P \cup \{Q_1\}, N \rangle_R$.

Such a direct dismissal of a discovered diagnosis $\mathcal{D}_i$ due to a newly added test case Q_j is indicated by $\text{\textcircled{k}} \checkmark_{(\mathcal{D}_i)} \overset{Q_j}{\Longrightarrow} \text{\textcircled{k}} \times$, i.e. the step number $\text{\textcircled{k}}$ at the shaft of the $\Longrightarrow$ is equal to the step number at the head of $\Longrightarrow$. In case of the invalidation of a *leading* diagnosis (i.e. one that was utilized in the computation of Q_j), on the contrary, the step number at the shaft is lower than the step number at the arrow head.

As shown at the top of Figure 11.3, the second query Q_2 computed from the leading diagnosis set $\mathbf{D}_\checkmark \cup \mathbf{D}_{calc} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_5\}$ is then answered by $u(Q_2) = true$ as well, wherefore the leading diagnoses $\mathcal{D}_2, \mathcal{D}_5$ are ruled out and added to $\mathbf{D}_\times$. So, the input argument $\mathbf{D}_\checkmark$ given to the next call of STATICHS in Algorithm 5 consists of the single diagnosis $\mathcal{D}_1$.

In the third iteration (see the picture given in Figure 11.3), STATICHS again executes in order to complete the leading diagnosis set to contain three elements. However, as we can say in advance, $\mathcal{D}_1$ is the only minimal diagnosis w.r.t. the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ which is also a diagnosis w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q_1, Q_2\}, N \rangle_R$. Nevertheless, STATICHS continues expanding the wpHS-tree until it has verified that this is the case ($\mathbf{Q} = []$). This is equivalent to finishing the construction of the non-interactive wpHS-tree that is generated by HS with parameters $n_{\min} = n_{\max} = \infty$. We want to stress that the construction of the entire wpHS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p() := p_\mathcal{K}()$ is inevitable in a debugging scenario where the (exact) solution to the Interactive Static KB Debugging problem is sought (the probability w.r.t. $p_\mathbf{D}()$ of a diagnosis can only be equal to 1 if there is only a single leading diagnosis returned by STATICHS).

In fact, there are five further diagnoses $\mathcal{D}_6, \dots, \mathcal{D}_{10}$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that are detected in iteration 3 and directly dismissed (added to $\mathbf{D}_\times$) after the validity check in line 10 of STATICHS. All other tree branches are closed due to the non-minimality (label $\times_{(\supset \mathcal{D}_i)}$) or duplicate criterion (label $\times_{(dup)}$). Due to $\sigma = 0$ and the associated necessity to grow the wpHS-tree until all leaf nodes are labeled, the final tree (19 labeled leaf nodes) depicted in Figure 11.3 is relatively large in comparison to the small size $|\mathcal{K}| = 7$.

This example might already give an idea of the potential explosion of the wpHS-tree produced by STATICHS in case the (exact) solution to the Interactive Static KB Debugging problem is desired. This is why it will usually make sense in practice to specify a fault tolerance $\sigma > 0$ which enables Algorithm 5 with $mode = static$ to escape from the generally intractable complexity of the complete investigation of all minimal diagnoses w.r.t. the input DPI (full construction of the wpHS-tree). However, in this concrete example, allowing a small fault tolerance σ has no effect either. Actually, $\sigma \geq 0.56$ is necessary to achieve a premature termination of the tree construction. This holds due to the fact that the probability distributions of leading diagnoses are $\langle p_\mathbf{D}(\mathcal{D}_1), p_\mathbf{D}(\mathcal{D}_2), p_\mathbf{D}(\mathcal{D}_3) \rangle = \langle 0.38, 0.37, 0.25 \rangle$ (after iteration 1)

and $\langle p_{\mathbf{D}}(\mathcal{D}_1), p_{\mathbf{D}}(\mathcal{D}_2), p_{\mathbf{D}}(\mathcal{D}_5) \rangle = \langle 0.44, 0.42, 0.14 \rangle$ (after iteration 2). Now, given say $\sigma := 0.6$, the stop criterion of Algorithm 5 would be met after iteration 2 because $p_{\mathbf{D}}(\mathcal{D}_{\max}) = p_{\mathbf{D}}(\mathcal{D}_1) = 0.44 \geq 0.4 = 1 - 0.6 = 1 - \sigma$. Nate that, in this case, the same (exact) solution would be returned as for the setting $\sigma := 0$. The (significant) difference is just that the final tree in this case has only 14 leaf nodes, of which only 7 are labeled (the labeling of a node is in general significantly more costly than the mere generation of a node). As opposed to this, the full tree comprises 19 labeled nodes. On the other side of the coin, choosing a value of $\sigma > 0.5$, for example, means that – from the point of view of the knowledge at the time Algorithm 5 terminates – a solution to Interactive Static KB Debugging is returned by Algorithm 5 which has a higher probability of not being the (exact) solution than of being the (exact) solution.

All in all, the execution of Algorithm 5 in this example performs

- 4 full QX calls, i.e. calls of QX that actually return a minimal conflict set (there are four minimal conflict sets labeled by C in the tree in Figure 11.3) and

- 20 validity checks, i.e. calls of QX that return 'no conflict' (one check for each of the 10 found minimal diagnoses; notice that QX does only perform a single KB validity check by ISKBVALID in case it returns 'no conflict', see Algorithm 1) or calls of ISKBVALID in line 10 in STATICHS (one call for each of the 10 found minimal diagnoses),

computes

- 10 minimal diagnoses w.r.t. the input DPI,

- 4 minimal conflict sets w.r.t. the input DPI and

- 2 queries and asks the user 2 logical formulas (1 per query)

and stores

- a maximum of 19 nodes (where node refers to the internal representation of a node in STATICHS as a set of edge labels along a path from the root node to a leaf node; there are even more nodes in the sense of tree nodes in the picture in Figure 11.3). □

$$\mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc} = \emptyset \cup \{\mathcal{D}_1, \mathcal{D}_2\} = \{[1], [2]\}$$
$$\mathbf{Q} = [\{5\}]$$
$$\mathbf{C}_{calc} = \{\langle 1, 2, 5 \rangle\}$$
$$\mathbf{D}_{\times} = \emptyset$$
$$\langle Q_1, \mathfrak{P}(Q_1) \rangle = \langle \{E \to \neg A\}, \langle \{\mathcal{D}_1\}, \{\mathcal{D}_2\}, \emptyset \rangle \rangle$$
$$u(Q_1) = false$$
$$\mathbf{D}_{\checkmark} = \{\mathcal{D}_2\}$$
$$\mathbf{D}_{out} = \mathbf{D}_{\times} = \{\mathcal{D}_1\}$$

$$\mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc} = \{\mathcal{D}_2\} \cup \{\mathcal{D}_3\} = \{[2], [5, 7]\}$$
$$\mathbf{Q} = []$$
$$\mathbf{C}_{calc} = \{\langle 1, 2, 5 \rangle, \langle 1, 2, 7 \rangle\}$$
$$\mathbf{D}_{\times} = \{\mathcal{D}_1\} = \{[1]\}$$
$$\langle Q_2, \mathfrak{P}(Q_2) \rangle = \langle \{Y \to \neg A\}, \langle \{\mathcal{D}_2\}, \{\mathcal{D}_3\}, \emptyset \rangle \rangle$$
$$u(Q_2) = false$$
$$\mathbf{D}_{\checkmark} = \{\mathcal{D}_3\}$$
$$\mathbf{D}_{out} = \{\mathcal{D}_2\}$$
$$\mathbf{D}_{\times} = \{\mathcal{D}_1, \mathcal{D}_2\}$$

$$\mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc} = \{\mathcal{D}_3\} \cup \emptyset = \{[5, 7]\}$$
$$\mathbf{Q} = []$$
$$\mathbf{C}_{calc} = \{\langle 1, 2, 5 \rangle, \langle 1, 2, 7 \rangle\}$$
$$\mathbf{D}_{\times} = \{\mathcal{D}_1, \mathcal{D}_2\} = \{[1], [2]\}$$
$$p_{\mathbf{D}}(\mathcal{D}_3) = 1$$
$$\Rightarrow \quad \text{return the solution KB } (\mathcal{K} \setminus \mathcal{D}_3) \quad \square$$

Figure 11.1: (Example 11.1) Solving the problem of Interactive Static KB Debugging (Problem Definition 6.2) for the example DPI given by Table 15.3 by means of Algorithm 5 and STATICHS.

$\mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc} = \emptyset \cup \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\} = \{[1, 4], [1, 6], [5, 4]\}, \quad \mathbf{Q} = \{\{5, 6\}, \{2, 4, 6\}, \{1, 2\}, \{2, 1\}, \{2, 3\}, \{5, 2\}, \{2, 4, 1\}, \{2, 4, 5\}, \{2, 4, 8\}\}, \quad \mathbf{D}_{\times} = \emptyset$

$\mathbf{C}_{calc} = \{\langle 1, 2, 5\rangle, \langle 2, 4, 6\rangle, \langle 1, 3, 4\rangle, \langle 1, 5, 6, 8\rangle\},$

$\langle Q_1, \mathfrak{P}(Q_1)\rangle = \{\{B \sqsubseteq K\}, \langle\{\mathcal{D}_1, \mathcal{D}_2\}, \{\mathcal{D}_3\}, \emptyset\rangle\rangle, \quad u(Q_1) = true, \quad \mathbf{D}_{\checkmark} = \{\mathcal{D}_1, \mathcal{D}_2\}, \quad \mathbf{D}_{out} = \mathbf{D}_{\times} = \{\mathcal{D}_3\}$
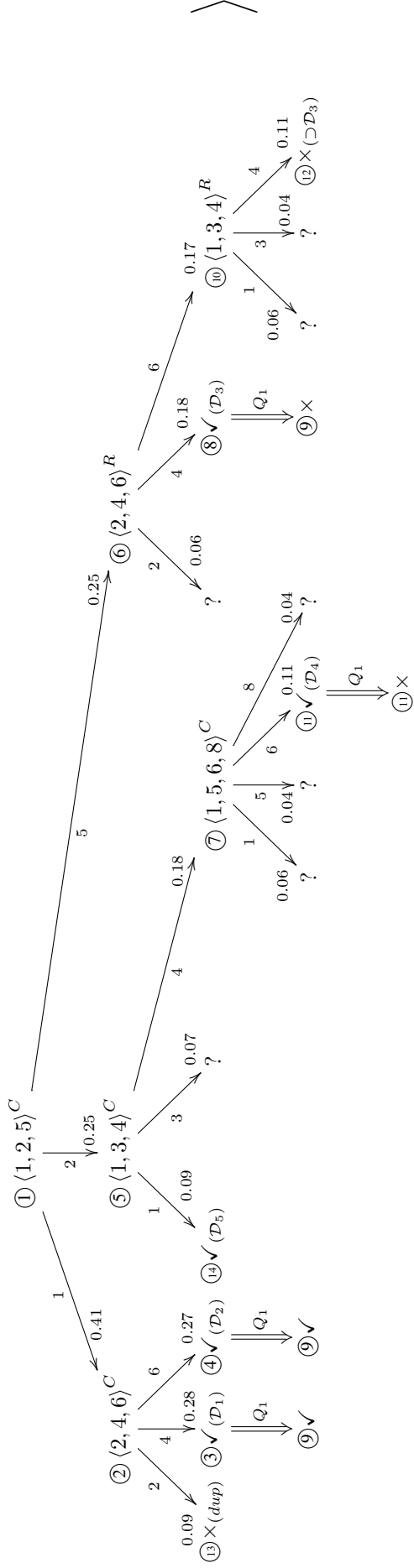
Figure 11.2: (Example 11.2) Solving the problem of Interactive Static KB Debugging (Problem Definition 6.2) for the example DPI given by Table 4.2 by means of Algorithm 5 and STATICHS.

$\mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc} = \{\mathcal{D}_1, \mathcal{D}_2\} \cup \{\mathcal{D}_5\} = \{[1,4],[1,6],[2,1]\}$, $\quad \mathbf{Q} = \{\{2,4,6\},\{2,3\},\{5,2\},\{2,4,1\},\{5,6,1\},\{5,6,3\}\}$, $\quad \mathbf{D}_{\checkmark} = \{\mathcal{D}_1\}$, $\quad \mathbf{D}_{out} = \{\mathcal{D}_2, \mathcal{D}_5\}$, $\quad \mathbf{D}_{\times} = \{\mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_2, \mathcal{D}_5\}$

$\mathbf{C}_{calc} = \{\langle 1,2,5\rangle, \langle 2,4,6\rangle, \langle 1,3,4\rangle, \langle 1,5,6,8\rangle\}$, $\quad \mathbf{D}_{\times} = \{\mathcal{D}_3, \mathcal{D}_4\} = \{[5,4],[2,4,6]\}$

$\langle Q_2, \mathfrak{P}(Q_2)\rangle = \langle \{B \sqsubseteq \exists r.F\}, \langle \{\mathcal{D}_1\}, \{\mathcal{D}_2, \mathcal{D}_5\}, \emptyset\rangle\rangle$, $\quad u(Q_2) = true$,

Iteration 3
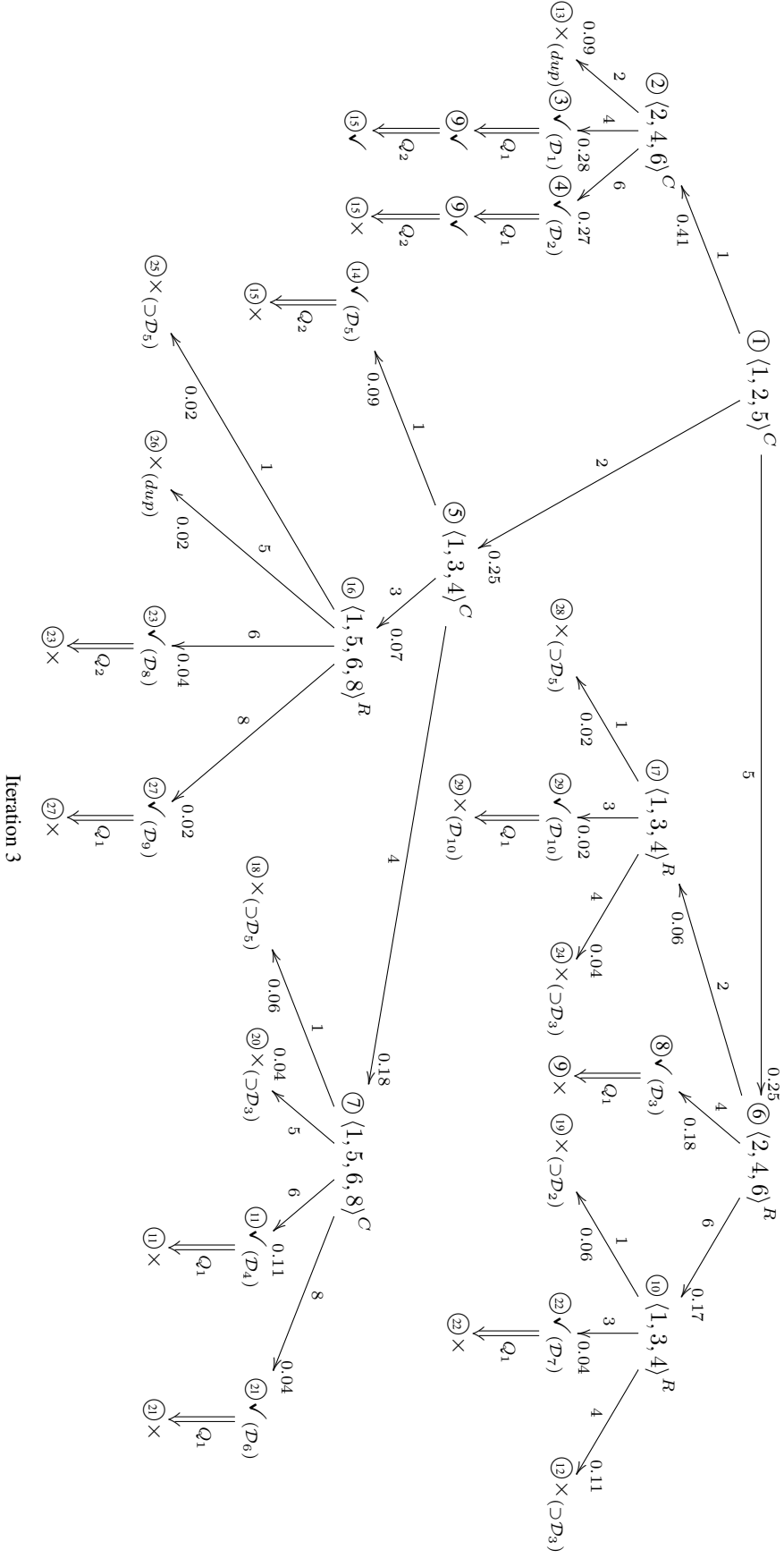
$\mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc} = \{\mathcal{D}_1\} \cup \emptyset = \{\mathcal{D}_1\}$, $\quad \mathbf{Q} = []$, $\quad \mathbf{C}_{calc} = \{\langle 1,2,5\rangle, \langle 2,4,6\rangle, \langle 1,3,4\rangle, \langle 1,5,6,8\rangle\}$,

$\mathbf{D}_{\times} = \{\mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_2, \mathcal{D}_5, \mathcal{D}_6, \mathcal{D}_7, \mathcal{D}_8, \mathcal{D}_9, \mathcal{D}_{10}\} = \{[5,4],[2,4,6],[1,6],[2,1],[2,4,8],[5,6,3],[2,3,6],[2,3,8],[5,2,3]\}$,

$p_D(\mathcal{D}_1) = 1 \quad \Rightarrow \quad$ return the solution KB $(\mathcal{K} \setminus \mathcal{D}_1) \cup p_1 \quad$ (p_1: cf. Table 4.2)
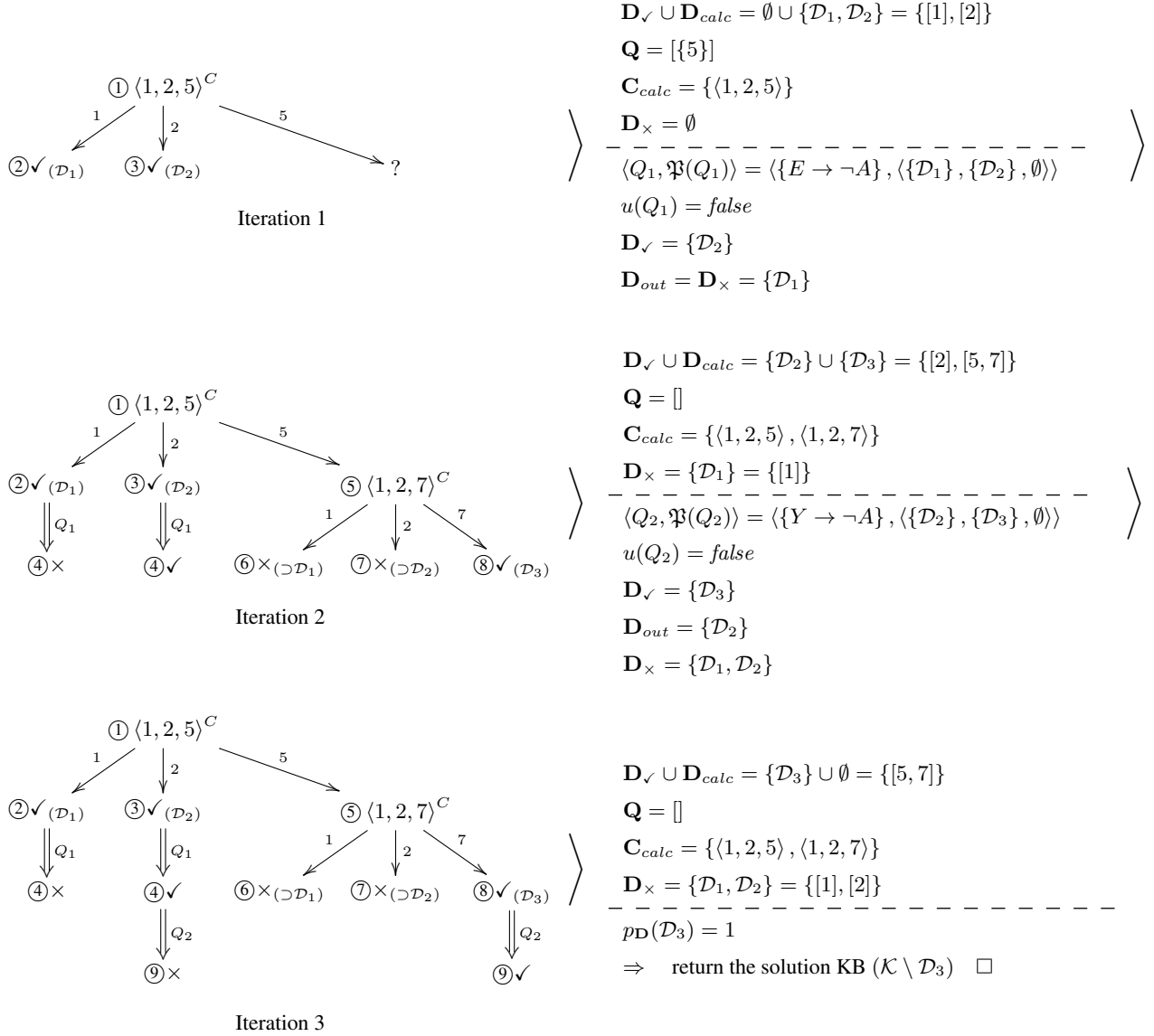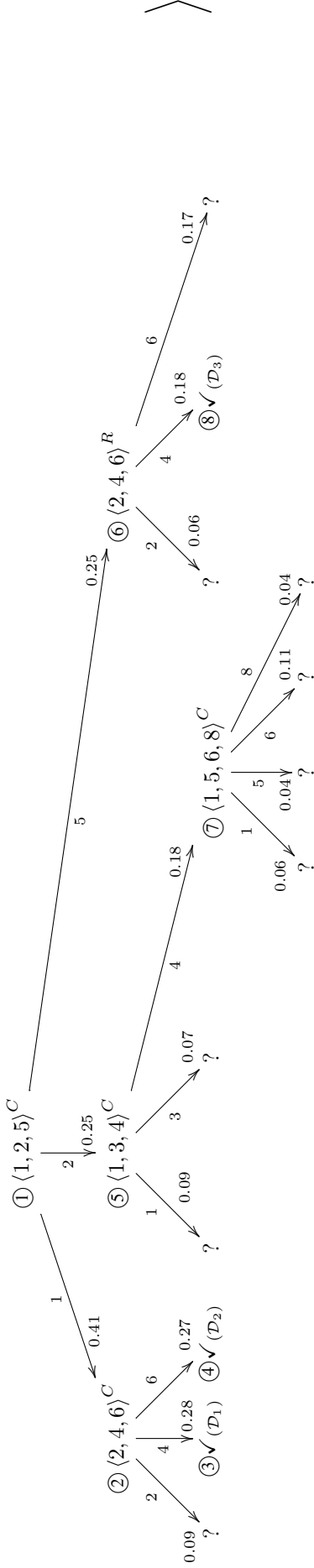
Figure 11.3: (Example 11.2 continued) Solving the problem of Interactive Static KB Debugging (Problem Definition 6.2) for the example DPI given by Table 4.2 by means of Algorithm 5 and STATICHS.

11.4 Correctness of the Algorithm

In this section we will demonstrate the correctness of STATICHS. That is, we will prove that STATICHS, given the inputs described in Algorithm 7, yields the outputs enumerated in Algorithm 7. Used in Algorithm 5 to iteratively compute a set of leading diagnoses for query generation, STATICHS in this way serves to solve the problem of Interactive Static KB Debugging approximately (parameter $\sigma > 0$ in Algorithm 5) or exactly ($\sigma = 0$).

After each call to STATICHS during Algorithm 5, the hitting set tree produced by STATICHS is a (partial) wpHS-tree w.r.t. the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given as an input to Algorithm 5 and $p_{nodes}()$ which can be directly obtained from the function $p()$ given as input to STATICHS. This proposition is made by Lemma 11.3.

In order to be able to prove this proposition, we formulate and prove two lemmata, Lemma 11.1 and 11.2. The former, which is given next, shows that this proposition holds for the very first call of STATICHS during the execution of Algorithm 5. The latter assures that this proposition holds for any further call of STATICHS during Algorithm 5 for an adequate set of input parameters to STATICHS. Finally, Lemma 11.3 exploits these results to ascertain that this proposition is satisfied for all calls of STATICHS.

Lemma 11.1. *Let the following be the input parameters to the* STATICHS *function:*

- $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *is the DPI given as input to Algorithm 5,*

- $n_{\min}, n_{\max}, t \in \mathbb{N}$ *where* $n_{\min} \geq 2$,

- *a function* $p : \mathcal{K} \to (0, 0.5)$,

- $\mathbf{Q} = [\emptyset]$,

- $P' = N' = \mathbf{D}_\times = \mathbf{D}_\checkmark = \mathbf{C}_{calc} = \emptyset$.

Then, STATICHS *creates a (partial) wpHS-tree T w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p_{nodes}()$ (cf. Definition 4.9) equivalent to one produced by Algorithm 2 with input parameters $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, $n_{\min}$, $n_{\max}$, t and $p()$ and returns $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_\times \rangle$ where $\langle \mathbf{D} \cup \mathbf{D}_\times, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ is the relevant data of T.*

Proof. Since all input parameters P', N', $\mathbf{D}_\times$, $\mathbf{D}_\checkmark$ and $\mathbf{C}_{calc}$ are equal to the empty set, $\mathbf{D}_{calc} = \emptyset$ and $\mathbf{Q}$ includes only the node $\emptyset$, we might regard $\langle \mathbf{D}_{calc}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ as the initial relevant data of some (partial) wpHS-tree which includes only an unlabeled root node. The root node $\emptyset$ cannot be labeled as otherwise it would be necessarily an element of $\mathbf{D}_{calc}$ if $\emptyset$ is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ or the set $\mathbf{C}_{calc}$ would include the conflict set that labels the root node.

$\mathbf{D}_\times$ can never be extended during the execution of STATICHS since line 13 can never be reached. This holds because the test made in line 10 can never be negative. Namely, as $P' = N' = \emptyset$, this test actually checks whether $\mathcal{K} \setminus$ node is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. Due to the fact that $L = valid$ has been output as a label for node (line 9) by the SLABEL function called in line 7, it must hold that $QX(\langle \mathcal{K} \setminus \text{node}, \mathcal{B}, P, N \rangle_R)$ yielded 'no conflict'. By Proposition 4.9, this implies that $\mathcal{K} \setminus$ node is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. Thence, $\mathbf{D}_\times = \emptyset$ definitely holds whenever STATICHS terminates.

Moreover, each node with the label $valid$ is added to $\mathbf{D}_{calc}$ since line 13 can never be reached. As a consequence, with the given input parameters, the execution of the code between line 2 and line 18 of Algorithm 7 has exactly the same effect as executing the code between line 2 and line 16 of Algorithm 2.

$\mathbf{D}_\checkmark$ can never be extended as there is no such modification operation at all in STATICHS. Thus, $\mathbf{D}_\checkmark = \emptyset$ holds throughout the execution of STATICHS.

Now, the SLABEL procedure is equivalent to the LABEL procedure of Algorithm 2, except for the first line of the non-minimality criterion. That is, in STATICHS (line 21) some nd is searched for in $\mathbf{D}_{(\times, \checkmark, calc)}$ whereas in Algorithm 2 (line 19) such nd is searched in $\mathbf{D}_{calc}$. However, we point out that

$\mathbf{D}_{(\times,\checkmark,calc)}$ in the SLABEL procedure corresponds to the set $\mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc}$ in STATICHS (cf. the call to SLABEL in line 7), where $\mathbf{D}_{\checkmark} = \mathbf{D}_{\times} = \emptyset$ is an invariant, as argued above. Taking these arguments into account, we have that $\mathbf{D}_{(\times,\checkmark,calc)}$ in SLABEL in line 21 is equal to $\mathbf{D}_{calc}$, just as in Algorithm 2.

Hence, with the given input parameters, we have verified that STATICHS acts equivalently to Algorithm 2. As Algorithm 2 produces a (partial) wpHS-tree T w.r.t. the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p_{nodes}()$ by Lemma 4.15, we infer that STATICHS also does so.

As opposed to Algorithm 2 which returns only $\mathbf{D}_{calc}$, STATICHS returns $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times} \rangle$ where $\mathbf{D} := \mathbf{D}_{calc} \cup \mathbf{D}_{\checkmark} = \mathbf{D}_{calc}$ since $\mathbf{D}_{\checkmark} = \emptyset$, as argued above. In that, $\mathbf{D}_{calc}$, $\mathbf{Q}$ and $\mathbf{C}_{calc}$ correspond exactly to the equally named collections in Algorithm 2 and $\mathbf{D}_{\times} = \emptyset$, as argued above. Therefore, by Corollary 4.6, $\langle \mathbf{D} \cup \mathbf{D}_{\times}, \mathbf{Q}, \mathbf{C}_{calc} \rangle = \langle \mathbf{D}_{calc}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ is the relevant data of the (partial) wpHS-tree T w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p_{nodes}()$ produced by Algorithm 2. $\qquad\square$

The next lemma manifests that STATICHS, given such parameters that $\langle \mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ is the relevant data of a (partial) wpHS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p_{nodes}()$, again yields a (partial) wpHS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p_{nodes}()$.

Lemma 11.2. *Let the following be the input parameters to the* STATICHS *function:*

- $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *is the DPI given as input to Algorithm 5,*

- P' *is the set of positive and* N' *is the set of negative test cases specified since the start of Algorithm 5 where* $P' \cup N' \supset \emptyset$,

- $n_{\min}, n_{\max}, t \in \mathbb{N}$ *where* $n_{\min} \geq 2$,

- *a function* $p : \mathcal{K} \to (0, 0.5)$,

- $\mathbf{D}_{\times} \neq \emptyset$, $\mathbf{D}_{\checkmark} \neq \emptyset$, $\mathbf{C}_{calc} \neq \emptyset$ *and* $\mathbf{Q}$ *such that* $\langle \mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ *is the relevant data of a (partial) wpHS-tree w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *and* $p_{nodes}()$ *produced by Algorithm 2 with input parameters* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *and* $p()$.

Then, STATICHS *creates a (partial) wpHS-tree T w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *and* $p_{nodes}()$ *equivalent to one produced by Algorithm 2 with input parameters* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *and* $p()$ *and returns* $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times} \rangle$ *where* $\langle \mathbf{D} \cup \mathbf{D}_{\times}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ *is the relevant data of T.*

Proof. Since $\langle \mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ is the relevant data of a (partial) wpHS-tree T w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p_{nodes}()$ produced by Algorithm 2 with input parameters $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p()$, it is clear that, if the construction of T is continued by an algorithm working equivalently to Algorithm 2 and using this relevant data, the relevant data of a (partial) wpHS-tree T' w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p_{nodes}()$ will be stored by this algorithm (Corollary 4.6). Therefore, we show that STATICHS is such an algorithm.

In Algorithm 2, the set of all already computed minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is denoted by $\mathbf{D}_{calc}$. Nodes labeled by $valid$ are added to $\mathbf{D}_{calc}$ (line 11) and $\mathbf{D}_{calc}$ is used in the non-minimality criterion in the LABEL function (line 19). If Algorithm 2 should be used to continue construction of T using the relevant data $\langle \mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$, the required setting is just to use $\mathbf{D}_{calc} := \mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark}$ and use $\mathbf{Q}$ and $\mathbf{C}_{calc}$ for the equally named variables in Algorithm 2. If then a new node nd labeled by $valid$ were added to $\mathbf{D}_{calc}$, we would have that $\mathbf{D}_{calc} := \mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark} \cup \{nd\}$. By Corollary 4.7, this set $\mathbf{D}_{calc}$ used by Algorithm 2 would at each point in time comprise exactly the $|\mathbf{D}_{calc}|$ most probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p_{nodes}()$.

In STATICHS, each node node labeled by $valid$ is added either to $\mathbf{D}_{calc}$, which is initially the empty set in STATICHS, or to $\mathbf{D}_{\times}$ (lines 11 and 13), i.e. node is added to $\mathbf{D}_{calc} \cup \mathbf{D}_{\times}$. Thus, it is also true to say that node is added to $\mathbf{D}_{calc} \cup \mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark}$. So, the first new node nd labeled by $valid$ is added to this set which is then equal to $\mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark} \cup \{nd\}$. This set is equal to the set $\mathbf{D}_{calc}$ that would be used by Algorithm 2 to further construct the (partial) wpHS-tree T.

In the non-minimality criterion in function SLABEL, $\mathbf{D}_{\times,\checkmark,calc}$ is used which is equal to the set $\mathbf{D}_{calc} \cup \mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark}$ in STATICHS (cf. the call to SLABEL in line 7). Hence, $\mathbf{D}_{calc} \cup \mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark}$ is used and modified in STATICHS in exactly the same way as $\mathbf{D}_{calc}$ is used and modified in Algorithm 2.

Apart from this, as can be easily verified, the labeling function SLABEL in STATICHS is identical to LABEL in Algorithm 2 and the way $\mathbf{Q}$ and $\mathbf{C}_{calc}$ are used and modified in STATICHS is exactly equivalent to the way these are used and modified in Algorithm 2.

What remains to be shown is that $\mathbf{D}_{calc} \cup \mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark}$, as $\mathbf{D}_{calc}$ in Algorithm 2, always contains all already computed minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ which are the $|\mathbf{D}_{calc} \cup \mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark}|$ most probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Since $\mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark}$ is the first set in the relevant data of a (partial) wpHS-tree T w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p_{nodes}()$ produced by Algorithm 2 with input parameters $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p()$, by Corollaries 4.6 and 4.7, it must be valid that $\mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark}$ comprises the $|\mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark}|$ most probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Since $\mathbf{D}_{calc}$ is initially defined to be the empty set in STATICHS, it is also true to say that $\mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc}$ comprises the $|\mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc}|$ most probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ when STATICHS starts executing. Since, by assumption, the same $p()$ is used by STATICHS as was used for the construction of the (partial) wpHS-tree T so far, the same ordering of $\mathbf{Q}$ is used by STATICHS as would be used by Algorithm 2 to further construct the (partial) wpHS-tree T. Therefore, $\mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc}$ must indeed comprise the $|\mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc}|$ most probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ at each point in time.

The set $\mathbf{D}$ in the tuple $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times} \rangle$ returned by STATICHS corresponds exactly to $\mathbf{D}_{calc} \cup \mathbf{D}_{\checkmark}$. So, $\mathbf{D} \cup \mathbf{D}_{\times} = \mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc}$.

To summarize, STATICHS acts exactly equivalently to Algorithm 2. As a consequence, Corollary 4.6 regarding Algorithm 2 applies to STATICHS as well. This means that the tuple consisting of the set of nodes labeled by $valid$, i.e. $\mathbf{D}_{\times} \cup \mathbf{D}_{\checkmark} \cup \mathbf{D}_{calc}$, the list of open nodes $\mathbf{Q}$ and the set of minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ in STATICHS store the relevant data of a (partial) wpHS-tree T as it could have been generated by Algorithm 2. This completes the proof. $\square$

Lemma 11.3. *Any call to* STATICHS *within Algorithm 5 yields an output* $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times} \rangle$ *where*

- $\langle \mathbf{D} \cup \mathbf{D}_{\times}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ *is the relevant data of T and*

- T *is a (partial) wpHS-tree w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *and* $p_{nodes}()$ *equivalent to one produced by Algorithm 2 with input parameters* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *and* $p()$.

Proof. As can be easily verified, the arguments given to STATICHS at the first time it is called throughout the execution of Algorithm 5 correspond exactly to the input parameters to STATICHS assumed in Lemma 11.1 (cf. the variable instantiations in lines 1-4 of Algorithm 5). Thus, by Lemma 11.1, we conclude that the first call to STATICHS during the runtime of Algorithm 5 yields the output $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times} \rangle$ where $\langle \mathbf{D} \cup \mathbf{D}_{\times}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ is the relevant data of T and T is a (partial) wpHS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p_{nodes}()$ equivalent to one produced by Algorithm 2 with input parameters $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p()$.

When this first call to STATICHS returns in Algorithm 5, $\mathbf{D}$ is renamed to become $\mathbf{D}_{\checkmark}$ in Algorithm 5 (line 8). $\mathbf{Q}$, $\mathbf{C}_{calc}$ and $\mathbf{D}_{\times}$ bear unmodified names within Algorithm 5. We point out that $\mathbf{Q}$ and $\mathbf{C}_{calc}$ are not modified anywhere in Algorithm 5. $\mathbf{D}_{\checkmark}$ and $\mathbf{D}_{\times}$ are modified only in lines 21 and 22. In these lines, a subset $\mathbf{D}_{out}$ of $\mathbf{D}_{\checkmark}$ is deleted from $\mathbf{D}_{\checkmark}$ and added to $\mathbf{D}_{\times}$.

$\mathbf{D}_{out}$ must be a subset of $\mathbf{D}_{\checkmark}$. This holds, first, because $\langle Q, \mathfrak{P}(Q) \rangle$ is a query Q w.r.t. the leading diagnoses $\mathbf{D}_{\checkmark}$ and the DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ together with its q-partition $\mathfrak{P}(Q)$ (CALCQUERY in line 16, cf. Section 9.2). Second, $\mathbf{D}_{out}$ corresponds either to $\mathbf{D}^+(Q)$ (if the answer $u(Q) = false$) or to $\mathbf{D}^-(Q)$ (if the answer $u(Q) = true$) where both sets must be subsets of the set of leading diagnoses $\mathbf{D}_{\checkmark}$ by Definition 7.2 (GETINVALIDDIAGS in line 19, cf. Section 9.2).

Hence, $\mathbf{D}_{\checkmark} \cup \mathbf{D}_{\times}$ remains unchanged throughout Algorithm 5. By the renaming of $\mathbf{D}$ to become $\mathbf{D}_{\checkmark}$ in Algorithm 5 (see the argumentation above), $\mathbf{D}_{\checkmark} \cup \mathbf{D}_{\times}$ is equal to the set $\mathbf{D} \cup \mathbf{D}_{\times}$ where

$\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times} \rangle$ is the output of the first call to STATICHS in Algorithm 5. Therefore, the relevant data $\langle \mathbf{D} \cup \mathbf{D}_{\times}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ of T is unmodified until the second call to STATICHS within Algorithm 5 is made.

So, we have that the arguments given to STATICHS at the second time it is called throughout the execution of Algorithm 5 correspond exactly to the input parameters to STATICHS assumed in Lemma 11.2. Notice that the probability measure $p_{\mathcal{K}}()$ which corresponds to the probability measure $p()$ in STATICHS is never changed throughout the while-loop in Algorithm 5 (cf. Section 9.2).

Thus, by Lemma 11.2, we conclude that the second call to STATICHS during the runtime of Algorithm 5 yields the output $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times} \rangle$ where $\langle \mathbf{D} \cup \mathbf{D}_{\times}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ is the relevant data of T' and T' is a (partial) wpHS-tree w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p_{nodes}()$ equivalent to one produced by Algorithm 2 with input parameters $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p()$.

By means of the same line of argument we used so far and further applications of Lemma 11.2 it can be derived that the proposition of this lemma holds for any call to STATICHS throughout Algorithm 5.   □

By means of the just proven Lemma 11.3, we are now able to show by the next lemma that STATICHS computes minimal diagnoses w.r.t. the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given as an input to Algorithm 5 in most-probable-first order. Further on, the next lemma will reveal that *only* minimal diagnoses w.r.t. the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ are computed by STATICHS which assures the soundness of STATICHS concerning the (input) DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. The soundness of STATICHS as regards the (current) DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ will be considered in Lemma 11.6 below.

Lemma 11.4. *Any call to* STATICHS *within Algorithm 5 yields an output* $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times} \rangle$ *where* $\mathbf{D} \cup \mathbf{D}_{\times}$ *is the set of* $|\mathbf{D} \cup \mathbf{D}_{\times}|$ *most probable (w.r.t.* $p_{nodes}()$*) minimal diagnoses w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Proof. Let T be the (partial) wpHS-tree T produced by any call to STATICHS within Algorithm 5. Then, by Lemma 11.3,

- T is equal to a (partial) wpHS-tree produced by Algorithm 2 with input parameters $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $p()$ and

- the first set $\mathbf{D}_{calc}$ in the relevant data $\langle \mathbf{D}_{calc}, \mathbf{Q}, \mathbf{C}_{calc} \rangle$ of T produced by Algorithm 2 corresponds to $\mathbf{D} \cup \mathbf{D}_{\times}$.

So, by Corollary 4.7, the proposition of this lemma follows.   □

Moreover, Lemma 11.3 provides the basis for showing the completeness of STATICHS. That is, Lemma 11.5 will manifest that *all* minimal diagnoses w.r.t. the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given as an input to Algorithm 5 will be found by STATICHS given that it keeps executing for a sufficiently long period of time.

Lemma 11.5. *Any call to* STATICHS *within Algorithm 5 where the execution of* STATICHS *terminates due to* $\mathbf{Q} = []$ *yields an output* $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times} \rangle$ *where* $\mathbf{D} \cup \mathbf{D}_{\times}$ *is the set of all minimal diagnoses w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Proof. The proposition of this lemma follows from Lemma 11.3 and Proposition 4.15 by an analogue argumentation as in the proof of Lemma 11.4.   □

The following lemma proves that STATICHS is sound w.r.t. the finding of minimal diagnoses w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$, i.e. the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given as an input to Algorithm 5 extended by all new positive and negative test cases P' and N', respectively, that have been collected so far.

Lemma 11.6. *If any call to* STATICHS *adds an element* $\mathcal{D}$ *to the set* $\mathbf{D}_{calc}$ *during the execution of Algorithm 5,* $\mathcal{D}$ *is a minimal diagnosis w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$.

Proof. By Lemma 11.4 we know that each node node that is added to $\mathbf{D}_{calc}$ by STATICHS is a minimal diagnosis w.r.t. the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Through the test for validity of $\mathcal{K} \setminus$ node w.r.t. $\langle \cdot, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ (cf. Definition 3.3) which must be successful before node is added to $\mathbf{D}_{calc}$ (ISKBVALID in line 10), we have that node must also be a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ by Proposition 3.2. Since node is a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ as argued and due to Proposition 12.4 (see page 200), there cannot be a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ which is a proper subset of node. Thence, node must be a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. $\quad\square$

We are now in a position to bring to proof that the first set $\mathbf{D}$ in the tuple output by any call of STATICHS in Algorithm 5 contains only these minimal diagnoses w.r.t. the (input) DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that are also minimal diagnoses w.r.t. the (current) DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. In other words, this means that the set of leading diagnoses used for query generation in Algorithm 5 consists only of minimal diagnoses w.r.t. the input DPI that are in agreement with the additional information given by all query answers so far.

Lemma 11.7. *Any call to* STATICHS *within Algorithm 5 yields an output* $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_\times \rangle$ *where* $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$.

Proof. The output set $\mathbf{D}$ of any call to STATICHS during the execution of Algorithm 5 corresponds to the set $\mathbf{D}_{calc} \cup \mathbf{D}_\checkmark$ in STATICHS. As per Lemma 11.6, $\mathbf{D}_{calc}$ includes only minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. By Lemma 11.4, $\mathbf{D}_{calc}$ includes only minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Therefore, we can conclude that $\mathbf{D}_{calc} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$. So, we must show that $\mathbf{D}_\checkmark \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ holds when any call to STATICHS during the execution of Algorithm 5 terminates. We will perform an induction proof.

Base Case: At the first call of STATICHS during the execution of Algorithm 5, the argument $\mathbf{D}_\checkmark$ passed to STATICHS is the empty set. As argued in the proof of Lemma 11.1, $\mathbf{D}_\checkmark$ is never modified throughout STATICHS. Thus, $\mathbf{D}_\checkmark = \emptyset \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ holds for the output of the first call to STATICHS. Therefore, the proposition of this lemma holds for the output of the first call of STATICHS.

Induction Step: Assume that the proposition of this lemma holds for the last-but-one call to STATICHS during the execution of Algorithm 5 (Induction Hypothesis). Consider the last, i.e. most recent, call to STATICHS during the execution of Algorithm 5.

First, the set $\mathbf{D}_\checkmark$ given as an input argument to STATICHS at the last call of STATICHS is unmodified throughout the entire execution of STATICHS, as already mentioned. Second, $\mathbf{D}_\checkmark = \mathbf{D}' \setminus \mathbf{D}_{out} \subseteq \mathbf{D}'$ holds where $\mathbf{D}'$ is the output of the last-but-one call of STATICHS by Algorithm 5 since the only modification to the set $\mathbf{D}'$ (which is denoted by $\mathbf{D}_\checkmark$ in Algorithm 5) during Algorithm 5 is the deletion (line 21) of exactly those diagnoses $\mathbf{D}_{out}$ in $\mathbf{D}'$ that are invalidated by the addition of the most recent test case (GETINVALIDDIAGS in line 19). That is, the input $\mathbf{D}_\checkmark$ to the most recent call to STATICHS includes only diagnoses that comply with the most recently added test case. Call the most recently added test case tc. By the Induction Hypothesis, $\mathbf{D}' \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup (P' \setminus \{tc\}), N \cup (N' \setminus \{tc\}) \rangle_R}$. Notice that either $tc \in P'$ or $tc \in N'$ holds, but not both. As $\mathbf{D}_\checkmark \subseteq \mathbf{D}'$, it must be true that $\mathbf{D}_\checkmark \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup (P' \setminus \{tc\}), N \cup (N' \setminus \{tc\}) \rangle_R}$ and $\mathbf{D}_\checkmark$ complies with the test case tc. Hence, we infer that $\mathbf{D}_\checkmark \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$. Consequently, the proposition of this lemma must hold for each call of STATICHS during the execution of Algorithm 5. $\quad\square$

The results proven so far in this section facilitate the proof of correctness of STATICHS:

Proposition 11.1 (Correctness of STATICHS)**.** *Any call to* STATICHS *(given the inputs described in Algorithm 7) within Algorithm 5 terminates and yields an output* $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_\times \rangle$ *where*

(1) it holds for $\mathbf{D}$ *that*

(a) $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ *is the set of most probable minimal diagnoses w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *that satisfy all test cases* P' *and* N' *such that*

    (i) $n_{\min} \leq |\mathbf{D}| \leq n_{\max}$ *and*

    (ii) $\mathbf{D} \supset \mathbf{D}_{\checkmark}$,

    if such a set $\mathbf{D}$ *exists, or*

(b) $\mathbf{D}$ *is equal to the set of all minimal diagnoses* $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$*, otherwise,*

where "most-probable" refers to the probability measure $p_{nodes}()$ *(cf. Definition 4.9) obtained from the given function* $p()$*;*

(2) $\mathbf{Q}$ *is the current queue of open (non-labeled) nodes of the produced (partial) wpHS-tree,*

(3) $\mathbf{C}_{calc}$ *is the set of all minimal conflict sets w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *computed so far and*

(4) $\mathbf{D}_{\times}$ *is the set of all minimal diagnoses w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *computed so far where each diagnosis in* $\mathbf{D}_{\times}$ *does not satisfy all test cases* P' *and* N'*.*

Proof. Termination of any call to STATICHS within Algorithm 5 is granted by the fact that each node is a subset of $\mathcal{K}$ wherefore $2^{|\mathcal{K}|}$ is a finite upper bound of the overall number of nodes that might be elements of $\mathbf{Q}$ during the execution of any call of STATICHS. Moreover, in each iteration of the repeat-loop in STATICHS, one element is removed from $\mathbf{Q}$ (line 6) and no once removed element can ever be readded to $\mathbf{Q}$. The latter is satisfied due to the non-minimality criterion (lines 21-23) that deletes all but one nodes set-equal to some set $X \subseteq \mathcal{K}$ before the first node set-equal to X is processed and due to the fact that no once labeled nodes, i.e. those nodes that are elements of $\mathbf{D}_{calc}$, $\mathbf{D}_{\checkmark}$ or $\mathbf{D}_{\times}$, are ever added to $\mathbf{Q}$ again (because there is no line of code in STATICHS that does so).

Proposition (1): During the execution of Algorithm 5 (and STATICHS), diagnoses are added to $\mathbf{D}_{\times}$ only in line 22. In this line, only and all diagnoses not complying with the most recent test case are added to $\mathbf{D}_{\times}$ (GETINVALIDDIAGS in line 19, cf. Section 9.2). Hence, no diagnosis in $\mathbf{D}_{\times}$ can be in $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$. Now, by Lemmata 11.4 and 11.7, we deduce that $\mathbf{D} \subset \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ is the set of most probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that satisfy all test cases P' and N'. If STATICHS does not terminate due to $\mathbf{Q} = []$, properties (a)-(i) and (a)-(ii) of $\mathbf{D}$ are direct consequences of the stop criterion in line 18 in STATICHS. Otherwise, we infer by Lemma 11.5 that (b) must be true.

Propositions (2) and (3) hold by Lemma 11.3 and the definition of relevant data of a (partial) wpHS-tree (cf. Remark 4.2).

Proposition (4): This proposition follows from the line of argument in the proof of proposition (1) above. □

Algorithm 7 Iterative Construction of a Static Hitting Set Tree

Input: a tuple $\langle \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R, \mathbf{Q}, t, n_{\min}, n_{\max}, \mathbf{C}_{calc}, \mathbf{D}_\checkmark, \mathbf{D}_\times, p(), P', N' \rangle$ consisting of

- the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given as input to Algorithm 5,

- the overall sets of positively (P') and negatively (N') answered queries added as test cases to $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ so far,

- the current queue $\mathbf{Q}$ of open (non-labeled) nodes of a (partial) wpHS-tree,

- some desired computation timeout t,

- a desired minimal ($n_{\min} \geq 2$) and maximal ($n_{\max}$) number of minimal diagnoses to be returned,

- the set $\mathbf{C}_{calc}$ of all minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ computed so far,

- the set $\mathbf{D}_\checkmark$ of all minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ computed so far that satisfy all test cases P' and N',

- the set $\mathbf{D}_\times$ of all minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ computed so far that do not satisfy all test cases P' and N'.

- a function $p : \mathcal{K} \to (0, 0.5)$.

Output: a tuple $\langle \mathbf{D}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_\times \rangle$ where

- $\mathbf{D}$ is the current set of leading diagnoses such that

  (a) $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ is the set of most probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that satisfy all test cases P' and N' such that
    (i) $n_{\min} \leq |\mathbf{D}| \leq n_{\max}$ and
    (ii) $\mathbf{D} \supset \mathbf{D}_\checkmark$,
    if such a set $\mathbf{D}$ exists, or

  (b) $\mathbf{D}$ is equal to the set of all minimal diagnoses $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \cap \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$, otherwise,

  where "most-probable" refers to the probability measure $p_{nodes}()$ (cf. Definition 4.9) obtained from the given function $p()$;

- $\mathbf{Q}$ is the current queue of open (non-labeled) nodes of the produced (partial) wpHS-tree,

- $\mathbf{C}_{calc}$ is the set of all minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ computed so far and

- $\mathbf{D}_\times$ comprises those minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ computed so far that do not satisfy all test cases P' and N'.

1: **procedure** STATICHS($\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R, \mathbf{Q}, t, n_{\min}, n_{\max}, \mathbf{C}_{calc}, \mathbf{D}_\checkmark, \mathbf{D}_\times, p(), P', N'$)
2:     $t_{start} \leftarrow$ GETTIME()
3:     $\mathbf{D}_{calc} \leftarrow \emptyset$
4:     **repeat**
5:         node $\leftarrow$ GETFIRST($\mathbf{Q}$)
6:         $\mathbf{Q} \leftarrow$ DELETEFIRST($\mathbf{Q}$)
7:         $\langle L, \mathbf{C} \rangle \leftarrow$ SLABEL($\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, node, $\mathbf{C}_{calc}, \mathbf{D}_\times \cup \mathbf{D}_\checkmark \cup \mathbf{D}_{calc}, \mathbf{Q}$)
8:         $\mathbf{C}_{calc} \leftarrow \mathbf{C}$
9:         **if** $L = valid$ **then**   ▷ node is minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$
10:             **if** ISKBVALID($\mathcal{K} \setminus$ node, $\langle \cdot, \mathcal{B}, P \cup P', N \cup N' \rangle_R$) **then**   ▷ ISKBVALID (see Algorithm 1)
11:                 $\mathbf{D}_{calc} \leftarrow \mathbf{D}_{calc} \cup \{$node$\}$ ▷ node does satisfy all test cases P' and N'
12:             **else**
13:                 $\mathbf{D}_\times \leftarrow \mathbf{D}_\times \cup \{$node$\}$ ▷ node does not satisfy all test cases P' and N'
14:         **else if** $L = closed$ **then**   ▷ do nothing, no need to store non-minimal diagnoses
15:         **else**   ▷ L must be a minimal conflict set
16:             **for** $e \in L$ **do**
17:                 $\mathbf{Q} \leftarrow$ INSERTSORTED(node $\cup \{e\}$, $\mathbf{Q}, p_{nodes}(), descending$)
18:     **until** $\mathbf{Q} = [] \vee [|\mathbf{D}_{calc}| \neq \emptyset \wedge |\mathbf{D}_{calc} \cup \mathbf{D}_\checkmark| \geq n_{\min} \wedge (|\mathbf{D}_{calc} \cup \mathbf{D}_\checkmark| = n_{\max} \vee$ GETTIME() $- t_{start} > t)]$
19:     **return** $\langle \mathbf{D}_{calc} \cup \mathbf{D}_\checkmark, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_\times \rangle$

20: **procedure** SLABEL($\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, node, $\mathbf{C}_{calc}, \mathbf{D}_{(\times, \checkmark, calc)}, \mathbf{Q}$)
21:     **for** nd $\in \mathbf{D}_{(\times, \checkmark, calc)}$ **do**
22:         **if** node $\supseteq$ nd **then**   ▷ node is a non-minimal diagnosis
23:             **return** $\langle closed, \mathbf{C}_{calc} \rangle$
24:     **for** nd $\in \mathbf{Q}$ **do**
25:         **if** node $=$ nd **then**   ▷ node is a duplicate node
26:             **return** $\langle closed, \mathbf{C}_{calc} \rangle$
27:     **for** $\mathcal{C} \in \mathbf{C}_{calc}$ **do**
28:         **if** $\mathcal{C} \cap$ node $= \emptyset$ **then**   ▷ the minimal conflict set $\mathcal{C}$ can be reused to label node
29:             **return** $\langle \mathcal{C}, \mathbf{C}_{calc} \rangle$
30:     $L \leftarrow$ QX($\langle \mathcal{K} \setminus$ node, $\mathcal{B}, P, N \rangle_R$)   ▷ see Algorithm 1 (page 48)
31:     **if** $L =$ 'no conflict' **then**   ▷ node is a diagnosis
32:         **return** $\langle valid, \mathbf{C}_{calc} \rangle$
33:     **else**   ▷ L is a *new* minimal conflict set ($\notin \mathbf{C}_{calc}$)
34:         $\mathbf{C}_{calc} \leftarrow \mathbf{C}_{calc} \cup \{L\}$
35:         **return** $\langle L, \mathbf{C}_{calc} \rangle$

Chapter 12

DYNAMICHS: A Dynamic Iterative Diagnosis Computation Algorithm

As the name already suggests, DYNAMICHS (Algorithm 8) is a procedure that solves the problem of *Interactive Dynamic KB Debugging* defined by Problem Definition 6.1 if used for leading diagnosis computation in Algorithm 5. DYNAMICHS is sound, complete and optimal w.r.t. the set of solutions of the *Interactive Dynamic KB Debugging* problem (this will be proven in Section 12.4.10). Optimality refers to the best-first computation of minimal diagnoses regarding a given probability measure.

12.1 Overview and Intuition

Synoptic View of the Algorithm.   DYNAMICHS (Algorithm 8) is employed as a subroutine in Algorithm 5 with $mode = dynamic$ to build up a hitting set tree iteratively. That is, each time DYNAMICHS is called in Algorithm 5, it expands the existing tree only to a sufficient extent in order to determine a desired number of new leading diagnoses used for the generation of the next query. Then, the leading diagnoses set is returned.

Outside of the DYNAMICHS method in Algorithm 5, a new diagnosis probability distribution is obtained by the diagnosis probability update (cf. Section 9.2). Once this distribution involves one diagnosis, the probability of which exceeds a predefined threshold $1 - \sigma$, the algorithm terminates. The output is a solution KB w.r.t. the *current DPI* built from this highly probable minimal diagnosis.

Remark 12.1   In case σ has a predefined value of zero, the output is the (exact) solution to the problem of *Interactive Dynamic KB Debugging* for the input DPI. In a scenario where some fault tolerance $\sigma > 0$ is given, the solution KB returned by Algorithm 5 is an approximation of the (exact) solution to *Interactive Dynamic KB Debugging* for the input DPI where a better approximation can be expected for smaller values of σ (cf. Remark 9.2). "Better" in this context refers to the satisfaction of desired semantic properties of the KB returned by Algorithm 5, i.e. desired entailments and desired non-entailments of the KB. The intuition is that specification of additional test cases T guarantees the output of a KB complying with these test cases, whereas accepting one – albeit highly probable – of multiple solution KBs without having incorporated T leaves open the possibility for this KB to not fulfill T.

However, answering queries is effort for an interacting user. Therefore, the approach that involves the "early" termination of the algorithm after a solution KB has a sufficiently high probability (lower than 1) constitutes a trade-off between exactness of the output and the effort of the user and overall execution time of the interactive KB debugging algorithm, respectively.   □

In case there is no highly probable leading diagnosis, a query constructed from the current set of leading diagnoses is asked to the user. The user's answer is incorporated into the current DPI resulting in a new DPI. Thereafter, DYNAMICHS is invoked again given this new DPI as an argument.

Storage of the Search Tree.   Between each two calls of DYNAMICHS in Algorithm 5, the "state" of the current hitting set tree is stored by variables

- $\mathbf{D}_{calc}$ – computed minimal diagnoses w.r.t. the current DPI,

- $\mathbf{Q}$ – the list of open, non-labeled nodes,

- $\mathbf{C}_{calc}$ – (not necessarily minimal) conflict sets w.r.t. the current DPI computed so far,

- $\mathbf{D}_{\supset}$ – non-minimal diagnoses w.r.t. the current DPI computed so far,

- $\mathbf{Q}_{dup}$ – non-labeled duplicate nodes (i.e. nodes corresponding to tree branches with the same set of edge labels as branches that are already present in the tree)

- $\mathbf{D}_{\times}$ – the empty set (is filled up during Algorithm 5 between two calls of DYNAMICHS with diagnoses from $\mathbf{D}_{calc}$ that have been invalidated by an answered query)

where nodes in the tree again store (among others) the edge labels on the path from the root node to themselves.

Search Tree Update.   It is immediately apparent from the enumeration given above that, in comparison to STATICHS, additional collections, i.e. $\mathbf{D}_{\supset}$ as well as $\mathbf{Q}_{dup}$, need to be maintained in order to "remember" the current tree while Algorithm 5 is processing outside of the method DYNAMICHS. The cause for these additional variables is the *tree update* necessary after each addition of a test case to a DPI. For, each iteration of DYNAMICHS considers a different DPI in terms of the test cases. And, any two different DPIs in general lead to a different hitting set tree and to different sets of minimal diagnoses and conflict sets. Hence, the idea of the tree update is the following: Reuse the partial hitting set tree T (stored by the variables described above) constructed before the new test case was added to the current DPI DPI_j and perform suitable modifications to T in order to obtain a tree T' such that the further expansion of T' allows to identify *all* minimal diagnoses w.r.t. the new DPI DPI_{j+1} resulting from the addition of the new test case to DPI_j. In other words, the tree update seeks to establish a tree that is equivalent to one built by execution of DYNAMICHS using the new DPI DPI_{j+1} starting from an empty tree.

Node Storage.   Notice that, unlike in STATICHS or HS, it is crucial to store nodes not as sets in DYNAMICHS, but as *ordered lists of formulas*. That is, each node nd stores a list of all the edge labels along the (directed) path in the hitting set tree from the root node to nd where the order of formulas in the list is given by the order of traversing the edge labels along this path. Additionally, DYNAMICHS stores the attribute nd.cs for each node nd which is an ordered list including the node labels, i.e. the conflict sets, along the path from the root node to nd in analogous way. Associating a node with these two lists instead of one set is necessary from the point of view of the tree update. Because this facilitates the differentiation between two nodes corresponding to an equal (partial) diagnosis. For example, there could be some node nd_1 that is "redundant" after some query Q has been answered, but there is a set-equal node nd_2 which is still "relevant" (set-equality refers to equal *sets*, not lists, of edge labels stored by two nodes). In this case, the algorithm should get rid of nd_1 (in order to save time and space) while preserving node nd_2 (in order to maintain completeness). Associating set-equal nodes with each other might thus either lead to unnecessary tree expansion steps (if none is deleted) or incompleteness of the algorithm concerning the consideration of all minimal diagnoses (in case both are deleted).

Addition of a Test Case Changes Set of Solutions. Unlike the STATICHS algorithm, which is strongly related to the non-interactive hitting set algorithm HS (Algorithm 2) as outlined in Section 11.1, the hitting set tree produced by DYNAMICHS will usually differ significantly from the non-interactive hitting set tree produced by HS. The reason for this is that in DYNAMICHS the initial DPI DPI_0 is not fixed (in that conflict sets and diagnoses are calculated only w.r.t. DPI_0), but *new test cases are also used for the computation of minimal conflict sets (and thus minimal diagnoses) and not only for the invalidation of diagnoses*. Hence, every time a query has been answered and a respective test case has been incorporated into the DPI, the minimal conflict sets computed for the old DPI DPI_j might not be minimal conflict sets w.r.t. the current DPI DPI_{j+1} anymore (see Examples 12.1 and 12.2). On the one hand, a minimal conflict set $\mathcal{C}$ w.r.t. DPI_j might be a non-minimal conflict set w.r.t. DPI_{j+1} (since there is a new minimal conflict set $\mathcal{C}' \subset \mathcal{C}$ w.r.t. DPI_{j+1}). On the other hand, there might be also "completely new" minimal conflict sets $\mathcal{C}_k$ w.r.t. DPI_{j+1} which are in no set-relationship with any minimal conflict set w.r.t. DPI_j.

Due to this changing set of minimal conflict sets, the set of minimal diagnoses is variable as well (cf. Proposition 4.6). To see this, let $\mathcal{D}$ be a minimal diagnosis w.r.t. DPI_j. Then $\mathcal{D}$ hits all minimal conflict sets $\mathcal{C}_k$ in $\mathbf{mC}_{DPI_j}$. Now, assume that $\mathcal{D}$ comprises (only) the element ax from $\mathcal{C}_k$, but there is a minimal conflict set $\mathcal{C}'_k$ in $\mathbf{mC}_{DPI_{j+1}}$ such that $\mathcal{C}'_k \subseteq \mathcal{C}_k \setminus \{ax\}$. In this case, $\mathcal{D}$ is not a (minimal) hitting set of all minimal conflict sets in $\mathbf{mC}_{DPI_{j+1}}$ (since $\mathcal{D}$ does not hit $\mathcal{C}'_k$), i.e. $\mathcal{D}$ is not a (minimal) diagnosis w.r.t. DPI_{j+1}. That means, $\mathcal{D}$ needs to be extended (by a hitting set of all minimal conflict sets in $\mathbf{mC}_{DPI_{j+1}}$ it does not hit) in order to become a diagnosis w.r.t. DPI_{j+1}. After extending $\mathcal{D}$, both situations might arise, either that $\mathcal{D}$ is a minimal diagnosis w.r.t. DPI_{j+1} or that $\mathcal{D}$ is a non-minimal diagnosis w.r.t. DPI_{j+1}. When the latter case occurs, DYNAMICHS might often be able to figure out that (the tree branch corresponding to) $\mathcal{D}$ is simply *redundant* (w.r.t. the new DPI DPI_{j+1}) and does not need to be considered during the further expansion of the hitting set tree (which searches for minimal diagnoses w.r.t. DPI_{j+1} and not w.r.t. DPI_j). That is, such redundant tree branches are unnecessary in order to explore all minimal diagnoses w.r.t. DPI_{j+1} (cf. Sections 12.1 and 12.4.5 for an explanation and precise characterization of redundancy).

As a consequence, the nice property of STATICHS that the set of minimal diagnoses that needs to be taken into account given DPI_{j+1} is a proper subset of the minimal diagnoses set that needed to be considered given DPI_j in no longer valid for DYNAMICHS. That is, the set of remaining solution candidates in DYNAMICHS is not guaranteed to "converge" *constantly* towards a singleton comprising only one solution. The DPI, the minimal conflict sets as well as the minimal diagnoses are "dynamic". What holds for both DYNAMICHS and STATICHS is the guarantee that the set of all (i.e. minimal and non-minimal) diagnoses is constantly shrinking, i.e. $\mathbf{aD}_{DPI_j} \supset \mathbf{aD}_{DPI_{j+1}}$ (as well will later prove by Corollary 12.4).

Search Tree Pruning. Let T be the hitting set tree produced in the j-th iteration of DYNAMICHS (i.e. T is the tree that was used to search for minimal diagnoses w.r.t. DPI_j). Then, after a new test case has been added to DPI_j, there are often redundant subtrees in T that can be pruned. The resulting tree T' can then be used in the $(j+1)$-th iteration of DYNAMICHS to identify minimal diagnoses w.r.t. the new DPI DPI_{j+1}. Using T instead of T' might lead to a significant time and (more severely) space overhead, due to the unnecessary expansion of redundant branches that are known to give no new information at all. Another approach could be to simply discard the entire tree T and start to construct a new one w.r.t. DPI_{j+1} from scratch. This strategy, however, will usually also suffer from a non-negligible time overhead since most of the tree T can be safely reused in iteration $j+1$ and only parts of it must be revised. In particular, this strategy would potentially involve many additional calls of QX (which internally calls an expensive reasoner) as, in the worst case (when no pruning is possible), the entire existing tree might be rebuilt.

As we shall see in Remark 12.5, Section 12.4 and Examples 12.1 as well as 12.2, the overhead in terms of (expensive) calls to a reasoner (i.e. calls of QX) due to tree pruning (compared to its impact on

the tree) is absolutely reasonable. In fact, only one call of a "fast version" of QX (see Section 12.4.6) might already lead to the deletion of 75% of the tree branches as one can see in the first pruning step in Example 12.2.

The evolution of the hitting set tree produced by Algorithm 5 using DYNAMICHS is thus characterized by *alternating expansion and pruning phases*. Also for very complex problems, in case that expansion phases are "short enough" such that tree pruning can take place "often enough", one might be able to keep the hitting set tree "small enough" to handle it efficiently. The extent of the expansion phase can be steered by the specification of the leading diagnosis parameters $n_{\min}$, $n_{\max}$ and t (cf. Section 9.2). In the extreme case, these can be defined in a way ($n_{\min} = n_{\max} = 2$) the algorithm will allow only the computation of a single further minimal diagnosis (in the first expansion phase: two diagnoses) before DYNAMICHS (i.e. the tree expansion phase) terminates and a further pruning phase might take place.

However, it is not automatically warranted that tree pruning is possible after each expansion phase. Similarly, no certainty is given that the transition from DPI_j to DPI_{j+1} just causes the deletion of parts of the tree and no additional expansion of the tree. In fact, this depends on certain properties of the test case that is added after an expansion phase (i.e. properties of the generated query).

Test Cases Affect Tree Pruning.   Some added test case might give rise to some pruning steps as well as it might induce the construction of new subtrees (where "new" means that these would be no subtress of a hitting set tree w.r.t. the previous DPI DPI_j). The latter situation occurs when "completely new" minimal conflict sets (see above) are introduced by the addition of a test case. If this is the only impact of a test case, then this test case has only a negative influence on the time and space complexity. In other words, none of the invalidated minimal diagnoses (and no other nodes in the tree) are redundant; but all of them must additionally hit the set of "completely new" minimal conflict sets (in order to become diagnoses w.r.t. DPI_{j+1}). Hence, in this case, the transition from DPI_j to DPI_{j+1} results only in monotonic growth of the tree. If possible, such "negative-impact test cases" must be avoided. On the other hand, one must strive for the usage of "positive-impact test cases", i.e. those that only trigger tree pruning, but no tree expansion. Defining and studying properties that constitute such "positive-impact test cases" and developing specialized algorithms for extracting exactly those types of queries that enable as substantial and effective pruning as possible is a topic of future research.

An idea pertinent to this issue could for example be to attempt to extract a query by means of the conflict set $\mathcal{C}$ that labels the root node of the tree. More concretely, if any answer to a query yields a new test case that leads to the introduction of a minimal conflict set that is a proper subset of $\mathcal{C}$, then it is for sure that significant pruning can take place (since *entire subtrees starting from the root of the tree* can be deleted). For instance, the first query Q_1 in Example 12.2 features this property. Roughly, the reasons for that are that Q_1 is an entailment of a proper subset $\mathcal{C}_{sub}$ of $\mathcal{C}$ (i.e. $\mathcal{C}_{sub}$ is a justification of Q_1, cf. Section 4.2) and Q_1 is "relevant" for this conflict set $\mathcal{C}$ to be a conflict set. In other words, the latter means that Q_1 can be used to "replace" the part $\mathcal{C}_{sub}$ of $\mathcal{C}$, i.e. $(\mathcal{C} \setminus \mathcal{C}_{sub}) \cup Q_1$ is invalid w.r.t. the given DPI. That is, addition of Q_1 to the positive test cases asserts the correctness of one part of $\mathcal{C}$, namely $\mathcal{C}_{sub}$ (cf. Example 12.2), wherefore the other part must be incorrect (because some part of a conflict set must be definitely incorrect). On the other hand, assignment of Q_1 to the negative test cases asserts exactly the incorrectness of $\mathcal{C}_{sub}$ wherefore the formulas $\mathcal{C} \setminus \mathcal{C}_{sub}$ become obsolete in the minimal conflict set $\mathcal{C}$ yielding the new minimal conflict set $\mathcal{C}' := \mathcal{C}_{sub}$. Another desirable property of Q_1 is that addition of Q_1 to either set of test cases does not imply the origination of any "completely new" conflict sets (see above) which result in additional growth of the tree.

That is, in its original form (without assuring only the usage of "positive-impact test cases"), the time and space complexity of DYNAMICHS is a function of the generated queries. There is a potential to perform significant pruning, but also the risk of significant tree growth. In case mostly "positive-impact queries" are generated and asked to the user, the performance might be very nice and significantly superior to the one of STATICHS. In the reverse case, the performance might be also worse than the one

of STATICHS. In the case of STATICHS, there is no chance for significant pruning, but also no chance for a tree growth that goes beyond the size of the non-interactive tree produced by HS.

In STATICHS, there are only expansion phases (in case the tree pruning described by Definition 4.8 is considered part of an expansion phase) which means that the tree constructed by STATICHS will constantly grow (apart from the deleted duplicate nodes and non-minimal diagnoses). All the user can do is hope that Algorithm 5 applying STATICHS will not run out of memory (cf. Section 11.1).

The idea is now to be able to use DYNAMICHS instead of STATICHS particularly if the latter runs out of memory soon. If the leading diagnosis parameters are specified small enough to prevent the hitting set tree produced during one expansion phase from becoming too large and test cases are not chosen unfavorably, the DYNAMICHS method should be able to outperform STATICHS significantly, as Examples 11.2 and 12.2 suggest.

12.2 Algorithm Walkthrough

Input Parameters. When DYNAMICHS (Algorithm 8) is called for the first time in Algorithm 5, the inputs $\mathbf{C}_{calc}$, $\mathbf{D}_{\checkmark}$, $\mathbf{D}_{\times}$, P' and N' correspond to the empty set and $\mathbf{Q} = [\emptyset]$ (cf. lines 1-4 and 10 in Algorithm 5). Further on, $\mathbf{D}_{calc}$ is defined to be the empty set at the beginning of each execution of DYNAMICHS. That is, DYNAMICHS starts the construction of the hitting set tree from an initial tree consisting of a single unlabeled root node $\emptyset$ ($\in \mathbf{Q}$). And, all collections that are later returned by DYNAMICHS in line 25, except for $\mathbf{Q}$, are initially empty. Further input arguments are the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ provided as an input to Algorithm 5, the sets of positively (P') and negatively (N') answered queries since the start of Algorithm 5 (both sets initially empty), the leading diagnosis computation parameters n_{min}, n_{max}, t (see description in Chapter 7 on page 95) and the probability measure $p() := p_{\mathcal{K}}()$ that assigns a probability in the interval $(0, 0.5)$ to each formula in $\mathcal{K}$ (see line 5 in Algorithm 5).

Tree Update during First Iteration of DYNAMICHS. Before the repeat-loop in DYNAMICHS is entered, the UPDATETREE function is called (line 4), but has no effect. This holds since UPDATETREE first iterates over all elements in $\mathbf{D}_{\times}$, then over all elements in $\mathbf{D}_{\supset}$ and finally over all elements in $\mathbf{D}_{\checkmark}$ where $\mathbf{D}_{\times} = \mathbf{D}_{\supset} = \mathbf{D}_{\checkmark} = \emptyset$, as pointed out before.

The Main Loop. During the repeat-loop, in each iteration the first node node in the queue $\mathbf{Q}$ of open (non-labeled) nodes is processed (GETFIRST, line 6). Notice that, anywhere throughout DYNAMICHS, nodes are added to $\mathbf{Q}$ in a way that a sorting of $\mathbf{Q}$ in descending order according to $p_{nodes}()$ (cf. Definition 4.9) is maintained (cf. INSERTSORTED in lines 17, 68, 77, 80, 100 and 103). Hence, the most probable node (according to $p_{nodes}()$) is always processed next.

So, when node is processed, it is first deleted from $\mathbf{Q}$ (DELETEFIRST, line 7). Then a test is performed whether node $\in \mathbf{D}_{\checkmark}$, i.e. whether node is already known to be a minimal diagnosis w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. In case this test is positive, node is directly added to $\mathbf{D}_{calc}$, the set of leading diagnoses that will be output by the current call of DYNAMICHS. Otherwise, the DLABEL function is called given node (i.a.) as a parameter (line 11).

Computation of a Node Label. The DLABEL function processes node as follows. First, the *non-minimality criterion* (lines 27-29) is checked. That is, among all nodes in $\mathbf{D}_{calc}$, one is searched which is a proper subset of node. If such a node nd is found, then node must be a non-minimal diagnosis w.r.t. the current DPI since, anytime throughout the execution of DYNAMICHS, $\mathbf{D}_{calc}$ contains only minimal diagnoses w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ (this will be proven later by Proposition 12.9). In this case, unlike in STATICHS, the branch in the hitting set tree corresponding to node cannot be simply discarded, but needs to be still stored (in the set $\mathbf{D}_{\supset}$). It is necessary to store non-minimal diagnoses as

these might become minimal diagnoses w.r.t. the new DPI obtained after the subsequent addition of a new test case to the current DPI (cf. Proposition 12.5).

In case the non-minimality criterion is not satisfied, the *reuse criterion* (lines 30-40) is checked next. That is, the set $\mathbf{C}_{calc}$ containing (not necessarily minimal) conflict sets w.r.t. the current DPI is browsed for a set $\mathcal{C}$ such that $\mathcal{C}$ and node are disjoint sets. If such a set $\mathcal{C}$ is found, there must be some set $X \subseteq \mathcal{C}$ which is a *minimal* conflict set w.r.t. the current DPI. This minimal conflict set X can then be used to label node since the set of edge labels along the path in the tree leading from the root node to node does not hit X (because it does not hit $\mathcal{C}$).

The minimality of $\mathcal{C}$ is verified by a call of $\mathrm{QX}(\langle \mathcal{C}, \mathcal{B}, P \cup P', N \cup N' \rangle_R)$ that yields X, a minimal conflict set w.r.t. the current DPI (cf. Proposition 4.9; notice that X must be a non-empty set due to Proposition 12.2, for details see Section 12.4). In case $X \subset \mathcal{C}$ (line 33), before X is returned as a label for node, the following tree pruning steps are performed:

- All the conflict sets $\mathcal{C}_i$ used as node labels in the hitting set tree or in duplicate tree branches so far (i.e. $\mathcal{C}_i \in$ nd.cs for a node nd $\in \mathbf{Q} \cup \mathbf{D}_\supset \cup \mathbf{Q}_{dup}$) such that $X \subset \mathcal{C}_i$ are replaced by X (PRUNEQDUP and PRUNE in lines 36-38),

- any subtree is pruned if its root node is linked to a node now labeled by X (replacing some $\mathcal{C}_i \supset X$) by an edge with label ax where ax is in $\mathcal{C}_i \setminus X$ (PRUNEQDUP and PRUNE in lines 36-38) and

- for each pruned node nd, if there is a non-pruned node in $\mathbf{Q}_{dup}$ suited to construct a node nd$'$ that can replace nd, nd$'$ is added to the collection of nodes from which nd was deleted (PRUNEQDUP and PRUNE in lines 36-38),

- all the conflict sets $\mathcal{C}_i \in \mathbf{C}_{calc}$ that are proper supersets of X are deleted from $\mathbf{C}_{calc}$ and X is added to $\mathbf{C}_{calc}$ (ADDSETDELSUPSETS in line 39).

Otherwise, $\mathcal{C}$ ($= X$) is directly returned by DLABEL without performing any tree pruning because the reused conflict set $\mathcal{C}$ is (still) a *minimal* conflict set w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ (notice that each element of $\mathbf{C}_{calc}$ was added to $\mathbf{C}_{calc}$ as a minimal conflict set w.r.t. some DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R$ where $P'' \subseteq P'$ and $N'' \subseteq N'$ during the execution of this or a previous call of DYNAMICHS). For an in-depth explanation of the pruning functions PRUNE and PRUNEQDUP the reader is kindly referred to Section 12.4.6.

Remark 12.2   During the execution of the first call of DYNAMICHS in Algorithm 5, no tree pruning can take place (neither within the scope of DLABEL nor anywhere else) since all elements of $\mathbf{C}_{calc}$ (initially the empty set) must be minimal conflict sets w.r.t. the input DPI which is at the same time the current DPI. Pruning of the hitting set tree is only possible in case some non-leaf nodes of the tree are labeled by conflict sets that are *not minimal* w.r.t. the current DPI. □

Given that the reuse criterion fails, QX is called given the *current* DPI $\langle \mathcal{K} \setminus$ node$, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ as an argument (line 41). If the output L is equal to 'no conflict', then we know by Proposition 4.9 that node is a diagnosis w.r.t. the current DPI, wherefore the label $valid$ is returned for node. Otherwise, the output L must be a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ that has an empty set-intersection with node. Since the reuse criterion failed, i.e. there is no set in $\mathbf{C}_{calc}$ that does not intersect with node, L must be a fresh minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ in the sense that $L \notin \mathbf{C}_{calc}$ must hold. Therefore the label L is first added to $\mathbf{C}_{calc}$ and then returned by DLABEL as a label for node.

Remark 12.3   Please notice that this call of QX to label a node is one of the key differences between STATICHS and DYNAMICHS. Whereas the former uses QX exclusively for the computation of minimal conflict sets w.r.t. the (static) input DPI exploiting just the initial sets of positive and negative test cases P and N, respectively, the latter employs QX to compute minimal conflict sets w.r.t. the (dynamic)

current DPI which includes all new test cases (P' and N') resulting from answered queries in the ongoing interactive debugging session so far. □

Processing of a Node Label. Back in the main procedure, the label L returned by the DLABEL function is processed as follows. If $L = valid$, then it is a fact that node is a minimal diagnosis w.r.t. the current DPI (cf. Proposition 12.9 in Section 12.4.9) wherefore node is added to the set $\mathbf{D}_{calc}$. Otherwise, if $nonmin$ is the returned label for node, node is added to the set $\mathbf{D}_{\supset}$ of non-minimal diagnoses w.r.t. the current DPI. Otherwise, i.e. if $L \notin \{valid, nonmin\}$, then L must be a minimal conflict set w.r.t. the current DPI (see the description of node label computation above). In this case, $|L|$ successor nodes of node are generated (lines 18 and 19). For each logical formula $e \in L$, a new node is computed from node (and node.cs) as $\mathsf{node}_e := \mathrm{ADD}(\mathsf{node}, e)$ and $\mathsf{node}_e.\mathsf{cs} := \mathrm{ADD}(\mathsf{node.cs}, L)$ which means that e is appended to the end of the list node and L is appended to the end of the list node.cs.

If there is already a node $\mathsf{nd} \in \mathbf{Q}$ such that $\mathsf{nd} = \mathsf{node}_e$ (line 20), where '=' applied to these lists means that the list nd *interpreted as a set* is equal to the list node_e *interpreted as a set* (cf. Section 12.4.1 for an explication of this notation), then there is already a branch in the existing tree which includes the same set of edge labels as the new node node_e. Note that the tree branch corresponding to nd will differ from the one corresponding to node_e in terms of the order of edge labels or (the order of) the node labels visited when traversed starting from the root node. As it makes no sense to expand two branches with equal sets of edge labels in a hitting set tree (cf. rule 6 in Definition 4.8) for time and space complexity reasons and the fact that the sought diagnoses are sets – and not lists – of edge labels in the tree, such a duplicate node node_e is stored in the separate list $\mathbf{Q}_{dup}$. This list $\mathbf{Q}_{dup}$ is always kept sorted by ascending node-cardinality (INSERTSORTED in line 21).

The purpose of storing and not deleting such nodes is the possibility that the now "active" branch nd might be pruned after the addition of some test case whereas node_e might be unaffected by that pruning step. In this case, node_e, given it meets certain properties (see Section 12.4 for details), can be reactivated and incorporated into the tree in order to replace nd. Had node_e just been discarded instead of being stored, the completeness of Algorithm 5 with $mode = dynamic$ would be violated in general. That is, we would not have any guarantee that all minimal diagnoses w.r.t. the current DPI are actually explored by the algorithm.

Otherwise, if there is no node in $\mathbf{Q}$ that is set-equal to node_e, then node_e is added to the k-th position in $\mathbf{Q}$ (INSERTSORTED in line 23) if there are (exactly) $k-1$ nodes in $\mathbf{Q}$ that have a probability as per $p_{nodes}()$ that is greater than or equal to $p_{nodes}(\mathsf{node}_e)$.

Stop Criterion. The repeat-loop of DYNAMICHS is executed until the stop criterion in line 24 is satisfied. The first criterion causing DYNAMICHS to terminate is $\mathbf{Q} = []$ which means that the complete hitting set tree has been constructed and no further nodes can be labeled. In this case, $\mathbf{D}_{calc}$ comprises all minimal diagnoses w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ (cf. Proposition 12.8).

If the first criterion is not met, then the second criterion is checked. That is, a test is performed which checks first whether there is at least one *new* diagnosis w.r.t. the current DPI in $\mathbf{D}_{calc}$ which was not returned by the last-but-one call of DYNAMICHS (i.e. which is not an element of $\mathbf{D}_{\checkmark}$). Notice that this criterion or $\mathbf{Q} = []$ will be definitely met after finite execution time of DYNAMICHS since either new nodes in $\mathbf{Q}$ will be processed (and labeled) until there is some new diagnosis w.r.t. the current DPI identified or the $\mathbf{Q}$ will become empty.

Additionally, the second criterion involves a test that checks whether the cardinality of $\mathbf{D}_{calc}$ amounts to at least $n_{\min}$ and either $|\mathbf{D}_{calc}| = n_{\max}$ or more than t time has passed since the start of the execution of DYNAMICHS. In the latter case, $n_{\min} \le |\mathbf{D}_{calc}| < n_{\max}$ holds. In the former case, $|\mathbf{D}_{calc}| = n_{\max}$ is satisfied.

Processing of the Leading Diagnoses Returned by DYNAMICHS. When a call of DYNAMICHS in Algorithm 5 returns $\langle \mathbf{D}_{calc}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_\times, \mathbf{D}_\supset, \mathbf{Q}_{dup} \rangle$, the set $\mathbf{D}_{calc}$ is stored in the variable $\mathbf{D}_\checkmark$ in Algorithm 5. Between two successive calls of DYNAMICHS in Algorithm 5, only this set $\mathbf{D}_\checkmark$ as well as $\mathbf{D}_\times$ are modified. The collections $\mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_\supset$ as well as $\mathbf{Q}_{dup}$ remain unchanged until they are used as input parameters when it comes to the next call of DYNAMICHS in Algorithm 5.

In case one diagnosis $\mathcal{D}_{\max}$ of the current leading diagnoses in $\mathbf{D}_\checkmark$ has a probability greater than or equal to $1 - \sigma$ as per the probability measure $p_{\mathbf{D}}()$ (see Section 9.2), the stop criterion of interactive KB debugging is met and the solution KB $(\mathcal{K} \setminus \mathcal{D}_{\max}) \cup U_{P \cup P'}$ w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ is returned to the user (GETSOLKB in line 14, cf. Section 9.2). Thereafter, Algorithm 5 terminates and no more calls of DYNAMICHS take place.

Otherwise, if no leading diagnosis satisfies the stop criterion, a query Q together with its q-partition $\mathfrak{P}(Q)$ is computed as has been detailed in Chapter 8 and Section 9.2. An answer $u(Q)$ to this query is submitted by the interacting user (line 17 in Algorithm 5). Then $u(Q)$ along with $\mathfrak{P}(Q)$ is exploited to figure out the subset $\mathbf{D}_{out}$ of $\mathbf{D}_\checkmark$ that does not comply with $u(Q)$. This set $\mathbf{D}_{out}$ is then deleted from $\mathbf{D}_\checkmark$ and added to $\mathbf{D}_\times$. Additionally, Q is added to the positive test cases P' if $u(Q) = \mathit{true}$ and to the negative test cases N' otherwise. Subsequently, DYNAMICHS is called again given

- the *updated parameters* $\mathbf{D}_\checkmark$, $\mathbf{D}_\times$, P' and N' (which are modified within and outside of DYNAMICHS during the execution of Algorithm 5),

- the *unchanged parameters* $\mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_\supset$ and $\mathbf{Q}_{dup}$ (which are modified only within DYNAMICHS during the execution of Algorithm 5) and

- the *constant parameters* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R, t, n_{\min}, n_{\max}$ and $p_{\mathcal{K}}()$ (which are not modified within or outside of DYNAMICHS during the execution of Algorithm 5).

The execution of this next and any subsequent call to DYNAMICHS runs in analogue way as described so far, except for the effect of the UPDATETREE function called at the very beginning of each execution of DYNAMICHS (recall that the execution of UPDATETREE had no effect during the *first* execution of DYNAMICHS). We shall now explicate how this function works in all other executions of DYNAMICHS, except for the first one.

Tree Update. Between line 48 and line 69, UPDATETREE goes through all nodes nd $\in \mathbf{D}_\times$ (recall that $\mathbf{D}_\times$ includes exactly these diagnoses that have been ruled out by the most recently answered query) and first performs the *Quick Redundancy Check* (QRC, lines 50-54) for nd. If the QRC is not successful, it additionally performs the *Complete Redundancy Check* (CRC, lines 56-60) for nd.

The QRC (for details see Lemma 12.6) aims at identifying whether nd is redundant and can be pruned, i.e. it attempts to find a witness of redundancy of nd. Informally, a *redundant node* in (redundant subtree of) the tree is a node (subtree) such that the further expansion of the current tree without this node (subtree) still yields to the detection of all minimal diagnoses w.r.t. the current DPI. A *witness of redundancy of* nd is a minimal conflict set $\mathcal{C}'$ w.r.t. the current DPI such that a superset $\mathcal{C} \supset \mathcal{C}'$ was used as a node label on the tree path nd represents (that is, there is some $i \le |\mathsf{nd.cs}|$ such that $\mathcal{C}$ is the i-th element of nd.cs, i.e. $\mathcal{C} = \mathsf{nd.cs}[i]$) and the label ($\mathsf{nd}[i]$) of the outgoing edge of $\mathcal{C}$ on the path represented by nd is an element not in $\mathcal{C}'$ (that is, an element in $\mathcal{C} \setminus \mathcal{C}'$). Formal and precise characterizations of redundancy of nodes and the witness of redundancy of a node are given by Definition 12.4 in Section 12.4.5.

To this end, the QRC involves the call of $\mathrm{QX}(\langle U_{\mathsf{nd.cs}} \setminus \mathsf{nd}, \mathcal{B}, P \cup P', N \cup N' \rangle_R)$ which returns X. If X is a set (and not 'no conflict'), then X is a minimal conflict set w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ (as $U_{\mathsf{nd.cs}} \setminus \mathsf{nd} \subseteq \mathcal{K}$, cf. Proposition 4.9). To check if X is in fact a witness of redundancy of nd, $X \subset \mathcal{C}$ (line 52) is tested for all $\mathcal{C} \in \mathsf{nd.cs}$. If such a $\mathcal{C}$ is located, X is a witness of redundancy of nd and the QRC is successful (expressed by $\mathit{quickRC} \leftarrow \mathit{true}$ in line 53). In this case, the execution is resumed at line 61.

The QRC bears its name due to the fact that it requires *at most one call of* QX (which internally performs expensive calls to a reasoner). Moreover, it passes to QX a (DPI including a) KB of a size that is generally significantly smaller than $|\mathcal{K}|$ where $|\mathcal{K}|$ is roughly the size of the KB used in the (more expensive) calls of QX made in the DLABEL function. Hence, the QRC will be usually very fast (cf. Proposition 4.8).

Otherwise, since the negative outcome of the QRC (which is sound, but not complete w.r.t. the finding of a witness of redundancy of nd) does not imply the non-existence of a witness of redundancy of nd, the CRC (for details see Lemma 12.7) must be performed. As the name already suggests, the CRC is sound *and complete* and will therefore be positive and yield a witness of redundancy if and only if there is some. The CRC involves multiple calls of $QX(\langle \mathsf{nd.cs}[i] \setminus \{\mathsf{nd}[i]\}, \mathcal{B}, P \cup P', N \cup N' \rangle_R)$, one for each conflict set nd.cs[i] in nd.cs. It is straightforward from the characterization of a witness of redundancy given before that, given the CRC returns a set X, X is a witness of redundancy of nd.

If nd is non-redundant, there cannot be any witness of redundancy of nd. Hence, the complete and sound method CRC will not find such a one. Therefore, $quickRC = false$ and $completeRC = false$ must hold in line 61. In this case, the for-loop in line 48 continues with the next node in $\mathbf{D}_\times$.

On the other hand, if nd is redundant, due to the completeness of CRC, either $quickRC = true$ or $completeRC = true$ must hold when it comes to the execution of the if-statement in line 61. At this point, it is guaranteed that the variable X stores a witness of redundancy of nd.

The CRC, contrary to the QRC, generally requires *multiple* (at most $|\mathsf{nd}|$) *calls of* QX (which internally performs expensive calls to a reasoner). But, like the QRC, it passes to QX a (DPI including a) KB of a size that is generally significantly smaller than $|\mathcal{K}|$. Furthermore, at most one call of QX will involve more than one call of ISKBVALID (see Algorithm 1), i.e. the function that calls the reasoner. This must be true since CRC only requires an additional call of QX if a witness of redundancy has not yet been found. And, each call of QX that does not find a witness of redundancy of nd returns 'no conflict' which necessitates only a single invocation of ISKBVALID. Hence, each execution of the CRC will be very fast in general as well (cf. Proposition 4.8).

What comes next is the pruning of all redundant nodes in the tree for which X is a witness of redundancy. Essentially, the same pruning steps are performed here as in the *reuse criterion* described in 'Computation of a node label' above. A detailed discussion of the pruning functions PRUNE as well as PRUNEQDUP can be found in Section 12.4.6.

Notice that a redundant node is guaranteed to be a redundant node in any further iteration of DYNAMICHS (using a new current DPI that incorporates new test cases). We will prove this by Lemma 12.4 in Section 12.4.5. So, nodes pruned by PRUNE or PRUNEQDUP can be deleted for good and do not need to be stored any longer. Moreover, it should be noted that only redundant nodes are pruned at any pruning step in DYNAMICHS. For, as long as a node in DYNAMICHS is not known to be redundant, some successor node of this node might be a minimal diagnosis w.r.t. the current DPI. Thus, the deletion of such a node could perhaps prevent the algorithm from finding a particular minimal diagnosis which would implicate the algorithm's incompleteness.

Remark 12.4 Since the removal of a node from a collection $S \in \{\mathbf{D}_\times, \mathbf{Q}, \mathbf{Q}_{dup}, \mathbf{D}_\supset\}$ within the scope of PRUNE or PRUNEQDUP can be followed by the re-addition to S of a suitable duplicate node constructed from a node stored in $\mathbf{Q}_{dup}$ (see Section 12.4.6 for a precise explanation of node replacements), $\mathbf{D}_\times$ might be changed both in that nodes are deleted from it and added to it during the for-loop (line 48). Therefore, the 'for nd $\in \mathbf{D}_\times$'-statement must be read as 'if nd is a node in the *current* set $\mathbf{D}_\times$ which has not yet been processed'. For a better code readability, we abstained from using a programmatically precise representation of this issue in Algorithm 9. □

Due to the soundness and completeness of QRC paired with CRC concerning the identification of a witness of redundancy for a given node and the accomplished pruning of (at least) all nodes in $\mathbf{D}_\times$ for which a witness of redundancy has been extracted, all nodes that are in $\mathbf{D}_\times$ when the algorithm reaches

line 67 are *non-redundant* nodes. Consequently, there is no evidence to exclude the remaining nodes in $\mathbf{D}_\times$ from the further search for minimal diagnoses. For this reason, each of these nodes is reinserted into $\mathbf{Q}$ by INSERTSORTED in line 68 such that the sorting of $\mathbf{Q}$ in descending order of $p_{nodes}()$ is maintained. Then these nodes are deleted from $\mathbf{D}_\times$. Thus, $\mathbf{D}_\times = \emptyset$ holds after each execution of UPDATETREE.

So, in DYNAMICHS, unlike in STATICHS, diagnoses (and nodes in general) are not ruled out due to the fact that they contradict an answered query, but only if they are (found to be) redundant. Nevertheless, a diagnosis that contradicts an answered query is a "hot candidate" for finding some witness of redundancy. For that reason, UPDATETREE searches for witnesses of redundancy (only) by means of $\mathbf{D}_\times$ which includes the most "suspicious" nodes. Namely, it comprises those nodes that were minimal diagnoses w.r.t. the last-but-one DPI, but have been invalidated by the most recently answered query. The two possible reasons for a diagnosis nd to be invalidated are its redundancy as defined above or that it does not hit a *new* minimal conflict set (which is not a subset of one in nd.cs) that has been introduced by the addition of the test case resulting from the user's query answer. Thus, it is likely to detect witnesses of redundancy by investigating nodes in $\mathbf{D}_\times$, as the QRC and the CRC do. Throughout the pruning steps performed in lines 62-65, witnesses of redundancy extracted from nodes in $\mathbf{D}_\times$ are exploited to remove redundant nodes in the other collections $\mathbf{Q}_{dup}$, $\mathbf{D}_\supset$ and $\mathbf{Q}$ as well.

Remark 12.5   It should be noted that the collections $\mathbf{Q}$ as well as $\mathbf{D}_\supset$ are not necessarily cleaned from all redundant nodes after all pruning steps in UPDATETREE are finished. At this point, all those redundant nodes are still elements of these collections for which no witness of redundancy was found (there might exist one, though) throughout the redundancy checks (QRC and CRC) performed.

Assuring the non-existence of redundant nodes in $\mathbf{Q}$ and $\mathbf{D}_\supset$ might involve extensive usage of the (expensive) reasoner. In the worst case, one call of QX for each non-leaf node along each path from the root node to a leaf node labeled by *nonmin* or to a leaf node that has no label would be necessary. However, the number of these non-leaf nodes is generally *exponential* in the maximum length of such a path in the tree. In comparison, the number of calls of QX for investigating all nodes in $\mathbf{D}_\times$ by QRC and CRC is *polynomial (linear)* in the maximum length of a tree path labeled by $\times$. For, the number of QX-calls cannot get larger than $(n_{\max} - 1)(|\mathsf{nd}_{\max}| + 1)$ where the constant $n_{\max}$ is the maximum number of desired leading diagnoses predefined by the user and $|\mathsf{nd}_{\max}|$ is the maximum cardinality of some $\mathsf{nd} \in \mathbf{D}_\times$. This holds since $|\mathbf{D}_\times| \leq n_{\max} - 1$ (cf. Corollary 7.3) and QRC requires at most one and CRC at most $|\mathsf{nd}_{\max}|$ QX-calls.

Other than that, the chance of locating new witnesses of redundancy by means of investigating nodes in $\mathbf{Q}$ and $\mathbf{D}_\supset$ can be assumed to be smaller than for nodes in $\mathbf{D}_\times$ since there is no indication or evidence that these nodes might be redundant. So, cleaning $\mathbf{Q}$ and $\mathbf{D}_\supset$ from all redundant nodes might be significant effort with negligible impact. Therefore, DYNAMICHS is designed to focus the search for witnesses of redundancy only on the "suspicious nodes" in $\mathbf{D}_\times$. $\square$

As mentioned above, when the execution arrives at line 70, only nodes that are definitely redundant (because they were deleted due to some witness of redundancy) have been deleted from the sets $\mathbf{Q}$, $\mathbf{D}_\times$, $\mathbf{D}_\supset$ and $\mathbf{Q}_{dup}$.

In lines 70-78, each node $\mathsf{nd} \in \mathbf{D}_\supset$ which has not been deleted throughout the pruning operations in line 65 is processed as follows: If there is no minimal diagnosis $\mathcal{D} \in \mathbf{D}_\checkmark$ such that $\mathsf{nd} \supset \mathcal{D}$, then nd is removed from $\mathbf{D}_\supset$ and reinserted into $\mathbf{Q}$ (lines 77 and 78) in a way the sorting of $\mathbf{Q}$ in descending order according to $p_{nodes}()$ is maintained (INSERTSORTED). This re-insertion is plausible since there is no more evidence of nd (which is a non-minimal diagnosis w.r.t. the last-but-one DPI) being a non-minimal diagnosis w.r.t. the current DPI (non-minimal diagnoses might become minimal diagnoses by the addition of test cases, cf. Section 12.4.3 and Proposition 12.5).

Otherwise, nd remains an element of the set of non-minimal diagnoses $\mathbf{D}_\supset$ w.r.t. the current DPI as $\mathbf{D}_\checkmark$ comprises exclusively minimal diagnoses w.r.t. the current DPI and one of these is a proper subset of nd.

In lines 79-80, all elements in $\mathbf{D}_\checkmark$, each of which is a minimal diagnosis w.r.t. the current DPI, are added to $\mathbf{Q}$ in a way the sorting of $\mathbf{Q}$ in descending order according to $p_{nodes}()$ is maintained.

Remark 12.6 Please notice that the elements of $\mathbf{D}_\checkmark$, although they are known to be minimal diagnoses w.r.t. the current DPI, are not directly added to the set of found leading diagnoses $\mathbf{D}_{calc}$ w.r.t. the current DPI, but to $\mathbf{Q}$. The reason for this is that there might be (not-yet-found) minimal diagnoses w.r.t. the current DPI (nodes in $\mathbf{Q}$ or successor nodes thereof) which were not minimal diagnoses w.r.t. the last-but-one DPI (and thus are no elements of $\mathbf{D}_\checkmark$) that have a higher probability as per $p_{nodes}()$ than elements of $\mathbf{D}_\checkmark$. For instance, such diagnoses might have been added to $\mathbf{Q}$ from the set $\mathbf{D}_\supset$ in line 77.

In this way, since always the first (and most probable) node in $\mathbf{Q}$ is processed next, a guarantee is given that $\mathbf{D}_{calc}$ always comprises the $|\mathbf{D}_{calc}|$ most probable minimal diagnoses w.r.t. the current DPI as per $p_{nodes}()$. The knowledge of the validity of minimal diagnoses in $\mathbf{D}_\checkmark$ w.r.t. the current DPI is however not forgotten, but exploited in line 12 (i.e. no call of DLABEL and QX is necessary for a node in $\mathbf{D}_\checkmark$ to be added to $\mathbf{D}_{calc}$), as elucidated in 'The main loop' above. $\square$

12.3 Illustrating Examples

In this section we will give two examples of how interactive KB debugging using DYNAMICHS (Algorithm 5 with parameter $mode = dynamic$) works. The first one will show the similarities and differences between the usage of DYNAMICHS (within Algorithm 5) and HS (within Algorithm 3) since it will depict the application of STATICHS on the same example DPI (see Table 15.3) that was used to show the functionality of HS in examples 4.8 and 4.9. At the same time, the first example will provide evidence that solving the problem of Interactive Dynamic KB Debugging can be less efficient than solving the problem of Interactive Static KB Debugging in terms of the number of query answers required from an interacting user. This will be discussed in more detail in Chapter 13.

The second example is supposed to deepen the reader's understanding of the way DYNAMICHS works. To this end, the example DPI provided by Table 4.2 will be used which constitutes a significantly harder (interactive) debugging task than the DPI investigated in the first example. This example will involve the construction of a relatively large hitting set tree in the first iteration of DYNAMICHS (which behaves very similarly to STATICHS as well as HS and constructs the same wpHS-tree as these methods), but will then show the power of the tree pruning that can be exploited in Interactive Dynamic KB Debugging in that the tree will shrink rapidly after the addition of test cases. Hence, this example will emphasize the advantage of the decision to search for a solution of Interactive Dynamic KB Debugging rather than for a solution of Interactive Static KB Debugging (more on that in Chapter 13).

Notice that, in the following examples, whenever some tuple or list occurs in an expression using set operators, it is interpreted as a set.

Example 12.1 In this example we assume that the author (called user throughout this example) of the (admissible) DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given by Table 15.3 applies Algorithm 5 with $mode = dynamic$ to interactively debug $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Further, the same scenario and parameter settings as in Example 11.1 are supposed. That is, $n_{\min} = n_{\max} = 2$ (notice that the time limit t is irrelevant in this case), $q := 1$ (cf. Chapter 8), $qsm()$ is equal to any query selection measure described in Section 9.3, $p_\mathcal{K}(ax) := c < 0.5$ for all $ax \in \mathcal{K}$, i.e. all formula fault probabilities are specified to be equal (to some constant c) and $\sigma := 0$.

The tree constructed and parameters computed and used by Algorithm 5 using DYNAMICHS are visualized by Figures 12.1 and 12.2. We use the same notation as in Figures 4.2, 4.3, 11.1, 11.2 and 11.3 which is described in Examples 4.8, 4.9, 11.1 and 11.2.

In the first iteration, i.e. during the execution of the first call of DYNAMICHS during Algorithm 5, the root node (initially the empty set) is labeled by the minimal conflict set $\langle 1, 2, 5 \rangle$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and three successor nodes, namely $\mathsf{nd}_1 := [1]$, $\mathsf{nd}_2 := [2]$ as well as $\mathsf{nd}_3 := [5]$ with $\mathsf{nd}_1.\mathsf{cs} = \mathsf{nd}_2.\mathsf{cs} =$

$\mathsf{nd}_3.\mathsf{cs} = [\langle 1, 2, 5 \rangle]$, are added to the queue of open nodes $\mathbf{Q}$. Since all formulas have been assigned an equal fault probability, DYNAMICHS conducts a breadth-first tree construction (as displayed by the numbers (i) that give the order of node labeling). That is, $\mathbf{Q}$ in this case is a first-in-first-out queue. In this vein, first $[1]$ and then $[2]$ are identified as minimal diagnoses w.r.t. the given DPI.

Since $\mathbf{D}_{calc} = \{[1], [2]\}$ has a cardinality of $n_{\min} = n_{\max} = 2$, the stop criterion of DYNAMICHS causes it to terminate and return $\langle \mathbf{D}_{calc}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{Q}, \mathbf{D}_\times, \mathbf{D}_\supset, \mathbf{Q}_{dup} \rangle = \langle \{[1], [2]\}, [[5]], \{\langle 1, 2, 5 \rangle\}, \emptyset, \emptyset, [] \rangle$, as shown in the upper right column in Figure 12.1.

Then, in Algorithm 5, outside of the DYNAMICHS procedure, the first query $Q_1 = \{E \to \neg A\}$ is computed from the leading diagnoses set $\{[1], [2]\}$. The q-partition $\mathfrak{P}(Q_1)$ associated with Q_1 is $\langle \{[1]\}, \{[2]\}, \emptyset \rangle$. The user's answer $u(Q_1)$ to Q_1 is then *false*. Thence, the set $\mathbf{D}_{out}$ is calculated from $\mathfrak{P}(Q_1)$ as $\mathbf{D}^+(Q_1) = \{[1]\}$ (due to negative answer, cf. Remark 7.4), deleted from $\mathbf{D}_\checkmark := \mathbf{D}_\checkmark \cup \mathbf{D}_{calc}$ to yield $\mathbf{D}_\checkmark = \{[2]\}$ and added to $\mathbf{D}_\times$ to yield $\mathbf{D}_\times = \{[1]\}$. Now, the set $\mathbf{D}_\checkmark$ corresponds to the set of all computed (i.e. added to $\mathbf{D}_{calc}$) minimal diagnoses w.r.t. the last-but-one DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that are minimal diagnoses w.r.t. current DPI $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q_1\} \rangle_R$, i.e. that satisfy the most recently answered query Q_1. The set $\mathbf{D}_\times$ comprises all computed (i.e. added to $\mathbf{D}_{calc}$) minimal diagnoses w.r.t. the last-but-one DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ that are not minimal diagnoses w.r.t. current DPI $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q_1\} \rangle_R$, i.e. that do not satisfy the most recently answered query Q_1.

These sets $\mathbf{D}_\checkmark$ and $\mathbf{D}_\times$ along with the collections $\mathbf{Q}, \mathbf{Q}_{dup}, \mathbf{D}_\supset$ and $\mathbf{C}_{calc}$ which are unmodified outside of DYNAMICHS are used as input arguments for the second call of DYNAMICHS. Notice that, in Figures 12.1 and 12.2, the resulting values of operations performed within DYNAMICHS are given in the righthand column above the dashed line whereas values computed outside of DYNAMICHS are given below the dashed line.

The execution of the second call of DYNAMICHS starts with a call of the UPDATETREE function. The purpose of this function is to transform the hitting set tree T that was constructed by the first call of DYNAMICHS into an updated hitting set tree T'. Whereas the tree T was used to locate minimal diagnoses w.r.t. the last-but-one DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, the modified tree T' should serve to generate minimal diagnoses w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q_1\} \rangle_R$. The parameters $\mathbf{D}_\checkmark$, $\mathbf{D}_\times$, $\mathbf{Q}$, $\mathbf{Q}_{dup}$, $\mathbf{D}_\supset$ and $\mathbf{C}_{calc}$ that represent the tree T (given at the top of the lefthand column in Figure 12.1), where $\mathbf{D}_\checkmark \cup \mathbf{D}_\times$ is equal to the set $\mathbf{D}_{calc}$ produced by the first call of DYNAMICHS, are i.a. given as input arguments to the UPDATETREE function.

As a first step within UPDATETREE, a redundancy check is performed for each diagnosis in $\mathbf{D}_\times$. In this case $\mathbf{D}_\times = \{\mathcal{D}_1\}$ since $\mathcal{D}_1$ is the only minimal diagnosis that has been ruled out by the most recently added negative test case Q_1. The purpose of the redundancy check is to figure out whether $\mathcal{D}_1$ is redundant w.r.t. the current DPI and must be pruned or whether it might be extended to become a minimal diagnosis w.r.t. the current DPI.

First, the Quick Redundancy Check (QRC) $\mathrm{QX}(\langle \{2, 5\}, \mathcal{B}, P, N \cup \{Q_1\} \rangle_R) = \langle 2, 5 \rangle$ (line 50 in DYNAMICHS) is executed for $\mathcal{D}_1$ which detects (line 52 in DYNAMICHS) that $\mathcal{D}_1$ (and possibly some further nodes) is redundant and can be pruned. This holds since the minimal conflict set $\langle 1, 2, 5 \rangle$ w.r.t. the last-but-one DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is not a minimal conflict set w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q_1\} \rangle_R$ because $\langle 2, 5 \rangle$ returned by QX is already a minimal conflict set w.r.t. the current DPI (cf. Proposition 4.9). We call the minimal conflict set $\langle 2, 5 \rangle$ a *witness of redundancy* for $\mathcal{D}_1$. Hence, all branches in the hitting set tree starting from the outgoing edge of $\langle 1, 2, 5 \rangle$ labeled by 1 can be safely deleted from all collections representing the new tree T' (warranted that all minimal diagnoses w.r.t. the current DPI can still be generated from the pruned tree T').

Please notice that the QRC involves only a single call of QX using a KB of a size (here: 2) that is generally significantly smaller than $|\mathcal{K}|$ (here: 7) which is roughly the size of the KB used in calls of QX made in the DLABEL function. Hence, the QRC will be usually very fast.

An illustration why $\langle 2, 5 \rangle$ "replaces" $\langle 1, 2, 5 \rangle$ as a minimal conflict set w.r.t. the current DPI can be given as follows: First, $\langle 1, 2, 5 \rangle$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ as it is a set-minimal subset

of $\mathcal{K}$ that entails $\{\neg A\} = n_1 \in N$, there is no other negative test case in N except for n_1 and there is no proper subset $\mathcal{C}'$ of $\langle 1, 2, 5 \rangle$ where $\mathcal{C}' \cup \mathcal{B} \cup U_P$ violates any $r \in R$ (see example 4.2 for a detailed explanation). Second, formula 2 implies in particular $E \to Y$ which, along with formula 5 ($Y \to \neg A$), yields $E \to \neg A$. As the negative answer to Q_1 is equivalent to postulating that $\{E \to \neg A\}$ must not be entailed by the KB desired by the user, we have that $\langle 2, 5 \rangle$ is a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q_1\} \rangle_R$. As neither $\{2\}$ nor $\{5\}$ is a invalid KB w.r.t. $\langle \cdot, \mathcal{B}, P, N \cup \{Q_1\} \rangle_R$ (cf. Corollary 4.1 and Definition 4.1), we have that $\langle 2, 5 \rangle$ is a *minimal* conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q_1\} \rangle_R$.

Because the QRC has been successful, yielding some witness of redundancy of $\mathcal{D}_1$, the Complete Redundancy Check (CRC) is no more necessary and the collections $\mathbf{Q}_{dup}$, $\mathbf{Q}$, $\mathbf{D}_\times$ as well as $\mathbf{D}_\supset$ are processed by the PRUNE and PRUNEQDUP functions, respectively, which involve the removal of all nodes in these collections that are redundant due to the witness $\langle 2, 5 \rangle$. In other words, all nodes are eliminated which correspond to a path in the tree that includes a node label $\mathcal{C}_{old} \supset \langle 2, 5 \rangle$ and the label e of the outgoing edge of $\mathcal{C}_{old}$ on this path is an element of $\mathcal{C}_{old} \setminus \langle 2, 5 \rangle$. Moreover, all the supersets of $\langle 2, 5 \rangle$ in $\mathbf{C}_{calc}$ (here, only $\langle 1, 2, 5 \rangle$) are replaced by $\langle 2, 5 \rangle$ since they are not minimal conflict sets anymore (ADDSETDELSUPSETS).

The pruning of nodes is expressed by dashed arrows in the pictures labeled by 'Updated Tree' in Figures 12.1 and 12.2 where the location of cutting a branch is marked by a crossline at the shaft of a dashed arrow. Furthermore, the elements of "old" minimal conflict sets that are no more elements of known (i.e. already computed) current minimal conflict sets are crossed out. As shown by the picture 'Updated Tree' in the righthand column of Figure 12.1, $\mathcal{D}_1$ is the only removed node during the pruning steps using the witness of redundancy $\langle 2, 5 \rangle$.

Since $\mathbf{D}_\supset = \emptyset$, UPDATETREE directly jumps to the last three lines where all elements of $\mathbf{D}_\checkmark$ are re-added to $\mathbf{Q}$ in sorted order (but at the same time remain elements of $\mathbf{D}_\checkmark$). In the figure, this is displayed by the $\overset{Q_1}{\Longrightarrow}$ pointing to a question mark (which stands for an open node) instead of a checkmark as in the case of the STATICHS algorithm. Notice that, although it is a fact that all elements of $\mathbf{D}_\checkmark$ are minimal diagnoses w.r.t. the current DPI, this step is necessary in order to make sure the set $\mathbf{D}_{calc}$ returned by any call of DYNAMICHS actually comprises the $|\mathbf{D}_{calc}|$ most probable minimal diagnoses w.r.t. the current DPI. For, there might be, for instance, some node that is a non-minimal diagnosis w.r.t. the last-but-one DPI (and is thus not an element of $\mathbf{D}_\checkmark$), but becomes a minimal diagnosis w.r.t. the current DPI and has a higher probability than some node in $\mathbf{D}_\checkmark$. Additionally, we want to point out that no calls of the DLABEL procedure are needed for diagnoses in $\mathbf{D}_\checkmark$ as we know their label must be *valid*. This is reflected by the test in line 8 in DYNAMICHS.

In the figure, all the updated collections $\mathbf{D}_\supset$, $\mathbf{C}_{calc}$, $\mathbf{Q}$ as well as $\mathbf{Q}_{dup}$, after being processed by UPDATETREE are shown at the bottom of fields labeled by UPDATETREE. We want to remark that $\mathbf{D}_\times$ is always the empty set at the end of the execution of UPDATETREE since each node in $\mathbf{D}_\times$ gets either pruned or is reinserted into $\mathbf{Q}$ as an open node. These updated collections represent the new pruned hitting set tree that can be further constructed in order to detect all and only minimal diagnoses w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q_1\} \rangle_R$. Note that the actions carried out by UPDATETREE take place between steps ④ and ⑤.

The expansion of this tree during the repeat-loop in DYNAMICHS is depicted by the picture named 'Iteration 2' in Figure 12.1. Namely, first (step ⑤) the node $[2]$ is directly labeled by *valid* (line 8) since it is a known minimal diagnosis w.r.t. the current DPI (as explained before). In the sixth step, $[5]$ is labeled by the minimal conflict set $\langle 1, 2, 7 \rangle$ w.r.t. the current DPI and three further nodes ($[5, 1]$, $[5, 2]$ and $[5, 7]$, all with nd.cs $= [\langle 2, 5 \rangle, \langle 1, 2, 7 \rangle]$) are generated as successor nodes of $[5]$ and are added to $\mathbf{Q}$. Now, $[5, 1]$ (first-in-first-out) is the foremost node in $\mathbf{Q}$ and is thus processed next and found to be a minimal diagnosis w.r.t. the current DPI. Therefore, DYNAMICHS terminates and returns i.a. the new set of leading diagnoses $\mathbf{D}_{calc} = \{[2], [5, 1]\}$.

Please notice the difference here to Example 11.1 where the node $\{5, 1\}$ never became part of $\mathbf{Q}$ in STATICHS due to the existence of a minimal diagnosis $[1]$ w.r.t. the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ which is

a proper subset of this node (and due to the fact that STATICHS must only consider minimal diagnoses *w.r.t. the input DPI*). In the current example, this node can only become relevant *w.r.t. the current DPI* if all (known) diagnoses (here, only [1]) that are proper subsets of it have already been pruned. It should now be clear to the reader why non-minimal nodes cannot be deleted for good as in STATICHS and why the set $\mathbf{D}_\supset$ is necessary in DYNAMICHS.

This leading diagnosis [5, 1] is also the reason why the second query $Q_2 = \{E \to G\}$ is different from the second query $(Y \to \neg A)$ calculated in Example 11.1.

The execution of the algorithm continues in an analogue manner as explained so far. In the following, we just want to explain some interesting aspects in the rest of its execution:

- After the query $Q_3 = \{Y \to \neg A\}$ (the same query as the second query in Example 11.1) is answered negatively and Q_3 is added to N' yielding the current DPI $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q_1, Q_2, Q_3\} \rangle_R$, the UPDATETREE function not only prunes $[2] = \mathcal{D}_2 \in \mathbf{D}_\times$ and adds $[5, 7] = \mathcal{D}_4 \in \mathbf{D}_\checkmark$ to $\mathbf{Q}$ as we delineated above for the first query Q_1, but adds $[5, 2] \in \mathbf{D}_\supset$ to $\mathbf{Q}$ as well. The reason for that is the deletion of the minimal diagnosis [2] w.r.t. the last-but-one DPI $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q_1, Q_2\} \rangle_R$ wherefore the last evidence for the non-minimality of node [5, 2] has been deleted. Hence, the status of [5, 2] as a non-minimal diagnosis is no more justified wherefore it must be added to the queue to preserve the completeness of the algorithm w.r.t. the finding of all minimal diagnoses w.r.t. the current DPI. And, indeed, [5, 2] is identified as minimal diagnosis ($\mathcal{D}_5$) in iteration 4.

- For each element of $\mathbf{D}_\times$ during each execution of UPDATETREE throughout the execution of Algorithm 5, the Quick Redundancy Check (QRC) is successful. That is, each witness of redundancy used for pruning throughout the entire runtime of the algorithm could be determined very fast. Namely, as it is easy to see from line 50 in DYNAMICHS, the KB used in the call of QX in the QRC for some node nd has a size in $O((|\mathsf{nd}| - 1)|\mathcal{C}_{\max}|)$ where $\mathcal{C}_{\max}$ is the minimal conflict set of maximum cardinality in $\mathbf{C}_{calc}$. In most of the cases, $|\mathsf{nd}| \ll |\mathcal{K}|$ as well as $|\mathcal{C}_{\max}| \ll |\mathcal{K}|$ will hold. The (usually more expensive) Complete Redundancy Check (CRC), which requires $O(|\mathsf{nd}|)$ calls to QX with a KB of size $O(|\mathcal{C}_{\max}| - 1)$, is thus never employed.

- In this example, the same minimal diagnosis [5, 7] is used to compute the finally returned solution KB as in Example 11.1. The only difference between both outputs is that the KB $(\mathcal{K} \setminus [5, 7]) \cup Q_4$ returned by DYNAMICHS in this example contains the new positive test case $Q_4 \in P'$. The output by STATICHS in Example 11.1 does not contain any newly specified positive test case in P' (cf. Remark 9.9), just the union of the "original" positive test cases in P (apart from that, there is not even a newly specified positive test case in Example 11.1).

- In spite of finding the same solution diagnosis, STATICHS requires fewer queries than DYNAMICHS. Notably, DYNAMICHS even needs a proper superset of the queries asked by STATICHS (Q_1, Q_2 in Example 11.1 are equal to Q_1, Q_3 in our current example) in this case. Such a proposition however cannot be made in general since the queries formulated by STATICHS generally differ from those formulated by DYNAMICHS. In this vein, it might just as well be the case that it takes DYNAMICHS fewer queries to finish than it takes STATICHS, due to its advantages in tree pruning.

All in all, the execution of Algorithm 5 in this example performs

- 2 full QX calls, i.e. calls of QX using the KB $\mathcal{K} \setminus \mathsf{node}$ for a node node that actually return a minimal conflict set (there are two minimal conflict sets labeled by C in Figures 12.1 and 12.2 which do not result from QRC, CRC or the minimality test of a conflict set in line 32 of DYNAMICHS),

- 4 fast QX calls, i.e. executions of QX within the scope of the QRC (one call of QX each for the QRC of $\mathcal{D}_1$, $\mathcal{D}_3$, $\mathcal{D}_2$ and $\mathcal{D}_5$),

- 5 validity checks, i.e. calls of QX that return 'no conflict' (one check for each of the five found minimal diagnoses where the identification of diagnoses $\mathcal{D}_2$ at step ⑤, $\mathcal{D}_2$ at step ⑨, $\mathcal{D}_4$ at step ⑭ and $\mathcal{D}_4$ at step ⑯ does not require any call to a reasoning service by means of $\mathbf{D}_{\checkmark}$, see line 8 in DYNAMICHS; notice that QX does only perform a single KB validity check by ISKBVALID in case it returns 'no conflict', see Algorithm 1) and

- 4 tree update processes involving 4 pruned nodes (1 per tree update),

computes

- 5 minimal diagnoses ($\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_4$ w.r.t. the input DPI and $\mathcal{D}_3$ and $\mathcal{D}_5$ w.r.t. some DPI resulting from the input DPI by addition of new test cases),

- 6 minimal conflict sets ($\langle 1, 2, 5 \rangle$ as well as $\langle 1, 2, 7 \rangle$ w.r.t. the input DPI and the subsets thereof $\langle 2, 5 \rangle$, $\langle 2, 7 \rangle$, $\langle 5 \rangle$ and $\langle 7 \rangle$ w.r.t. some DPI resulting from the input DPI by addition of new test cases) and

- 4 queries and asks the user 4 logical formulas (1 per query)

and stores

- a maximum of 4 nodes (where node refers to the internal representation of a node nd in DYNAMICHS as a list of edge labels (nd) and a list of node labels (nd.cs) along a path from the root node to a leaf node). □
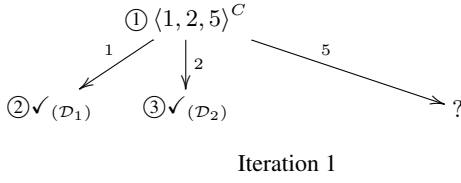
$$\mathbf{D}_{calc} = \{\mathcal{D}_1, \mathcal{D}_2\} = \{[1], [2]\}$$

$$\mathbf{Q} = [[5]]$$

$$\mathbf{C}_{calc} = \{\langle 1, 2, 5 \rangle\}$$

$$\mathbf{D}_{\supset} = \emptyset$$

$$\mathbf{Q}_{dup} = []$$

$$\langle Q_1, \mathfrak{P}(Q_1) \rangle = \langle \{E \to \neg A\}, \langle \{\mathcal{D}_1\}, \{\mathcal{D}_2\}, \emptyset \rangle \rangle$$

$$u(Q_1) = false$$

$$\mathbf{D}_{\checkmark} = \{\mathcal{D}_2\}, \mathbf{D}_{\times} = \{\mathcal{D}_1\}$$

Iteration 1

Updated Tree

UPDATETREE:

QRC $(\mathcal{D}_1)$: $\mathbf{QX}(\langle \{2, 5\}, \mathcal{B}, P, N \cup \{Q_1\} \rangle) = \langle 2, 5 \rangle$

$\Rightarrow$ PRUNE: $\langle 1, 2, 5 \rangle \to \langle 2, 5 \rangle$

- prune all subtrees starting from nodes $\langle 1, 2, 5 \rangle$ by outgoing edge with label 1
- replace by $\langle 2, 5 \rangle$ all node labels in the tree that are proper supersets of $\langle 2, 5 \rangle$

$\Rightarrow$ $\mathbf{D}_{\supset} = \emptyset$, $\mathbf{C}_{calc} = \{\langle 2, 5 \rangle\}$,

$\mathbf{Q} = [[2], [5]]$, $\mathbf{Q}_{dup} = []$,
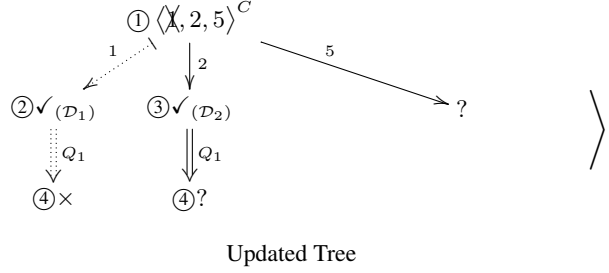
$$\mathbf{D}_{calc} = \{\mathcal{D}_2, \mathcal{D}_3\} = \{[2], [5, 1]\}$$

$$\mathbf{Q} = [[5, 2], [5, 7]]$$

$$\mathbf{C}_{calc} = \{\langle 2, 5 \rangle, \langle 1, 2, 7 \rangle\}$$

$$\mathbf{D}_{\supset} = \emptyset$$

$$\mathbf{Q}_{dup} = []$$

$$\langle Q_2, \mathfrak{P}(Q_2) \rangle = \langle \{E \to G\}, \langle \{\mathcal{D}_3\}, \{\mathcal{D}_2\}, \emptyset \rangle \rangle$$

$$u(Q_2) = false$$

$$\mathbf{D}_{\checkmark} = \{\mathcal{D}_2\}, \mathbf{D}_{\times} = \{\mathcal{D}_3\}$$
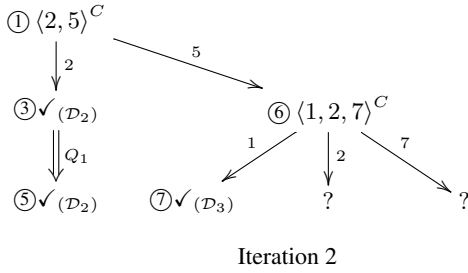
Iteration 2

UPDATETREE:

QRC $(\mathcal{D}_3)$: $\mathbf{QX}(\langle \{2, 7\}, \mathcal{B}, P, N \cup \{Q_1, Q_2\} \rangle) = \langle 2, 7 \rangle$

$\Rightarrow$ PRUNE: $\langle 1, 2, 7 \rangle \to \langle 2, 7 \rangle$

$\Rightarrow$ $\mathbf{D}_{\supset} = \emptyset$, $\mathbf{C}_{calc} = \{\langle 2, 5 \rangle, \langle 2, 7 \rangle\}$

$\mathbf{Q} = [[2], [5, 2], [5, 7]]$, $\mathbf{Q}_{dup} = []$
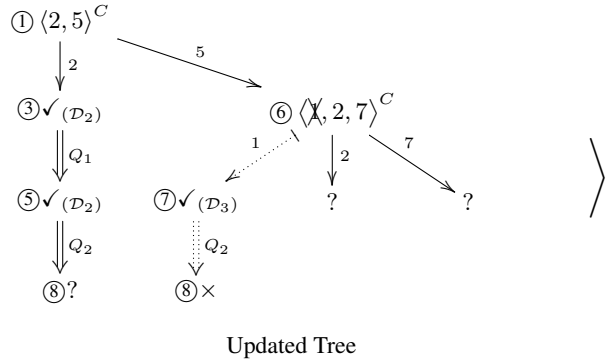
Updated Tree

$$\mathbf{D}_{calc} = \{\mathcal{D}_2, \mathcal{D}_4\} = \{[2], [5, 7]\}$$

$$\mathbf{Q} = []$$

$$\mathbf{C}_{calc} = \{\langle 2, 5 \rangle, \langle 2, 7 \rangle\}$$

$$\mathbf{D}_{\supset} = \{[5, 2]\}$$

$$\mathbf{Q}_{dup} = []$$

$$\langle Q_3, \mathfrak{P}(Q_3) \rangle = \langle \{Y \to \neg A\}, \langle \{\mathcal{D}_2\}, \{\mathcal{D}_4\}, \emptyset \rangle \rangle$$

$$u(Q_3) = false$$

$$\mathbf{D}_{\checkmark} = \{\mathcal{D}_4\}, \mathbf{D}_{\times} = \{\mathcal{D}_2\}$$
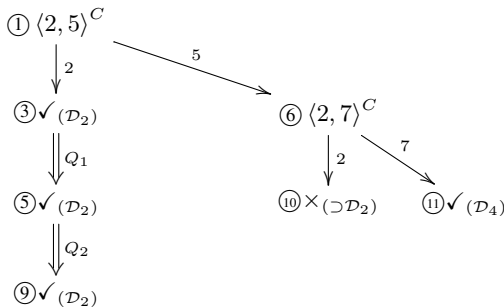
Iteration 3

Figure 12.1: (Example 12.1) Solving the problem of Interactive Dynamic KB Debugging (Problem Definition 6.1) for the example DPI given by Table 15.3 by means of Algorithm 5 and DYNAMICHS.

UPDATETREE:

QRC $(\mathcal{D}_2)$: $\mathrm{QX}(\langle\{5\}, \mathcal{B}, P, N \cup \{Q_1, Q_2, Q_3\}\rangle) = \langle 5\rangle$

$\Rightarrow$ PRUNE: $\langle 2, 5\rangle \rightarrow \langle 5\rangle$

$\Rightarrow \mathbf{D}_\supset = \emptyset, \mathbf{C}_{calc} = \{\langle 5\rangle, \langle 2, 7\rangle\}$

$\quad \mathbf{Q} = [[5, 2], [5, 7]], \mathbf{Q}_{dup} = []$
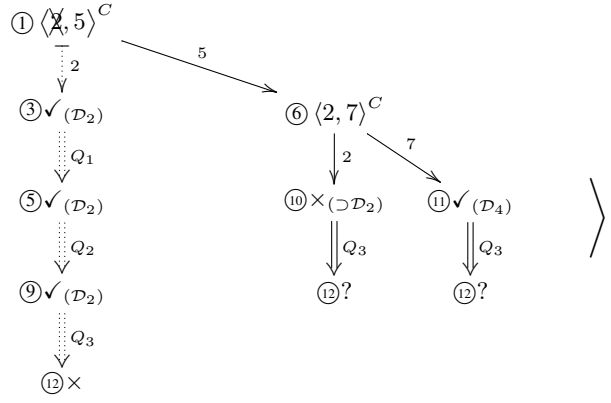
Updated Tree

Iteration 4

$\mathbf{D}_{calc} = \{\mathcal{D}_4, \mathcal{D}_5\} = \{[5, 7], [5, 2]\}$

$\mathbf{Q} = []$

$\mathbf{C}_{calc} = \{\langle 5\rangle, \langle 2, 7\rangle\}$

$\mathbf{D}_\supset = \emptyset$

$\mathbf{Q}_{dup} = []$

$\langle Q_4, \mathfrak{P}(Q_4)\rangle = \langle\{E \rightarrow Z\}, \langle\{\mathcal{D}_4\}, \{\mathcal{D}_5\}, \emptyset\rangle\rangle$

$u(Q_4) = true$

$\mathbf{D}_\checkmark = \{\mathcal{D}_4\}, \mathbf{D}_\times = \{\mathcal{D}_5\}$
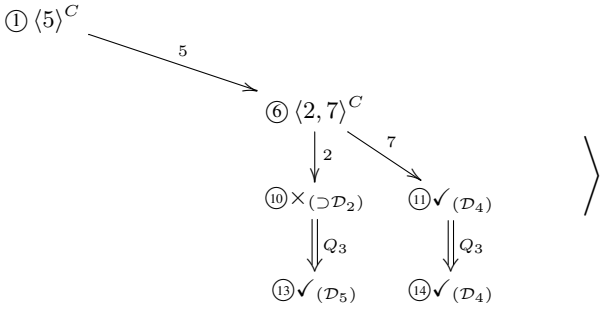
UPDATETREE:

QRC $(\mathcal{D}_5)$:

$\mathrm{QX}(\langle\{7\}, \mathcal{B}, P \cup \{Q_4\}, N \cup \{Q_1, Q_2, Q_3\}\rangle) = \langle 7\rangle$

$\Rightarrow$ PRUNE: $\langle 2, 7\rangle \rightarrow \langle 7\rangle$

$\Rightarrow \mathbf{D}_\supset = \emptyset, \mathbf{C}_{calc} = \{\langle 5\rangle, \langle 7\rangle\}$

$\quad \mathbf{Q} = [[5, 7]], \mathbf{Q}_{dup} = []$
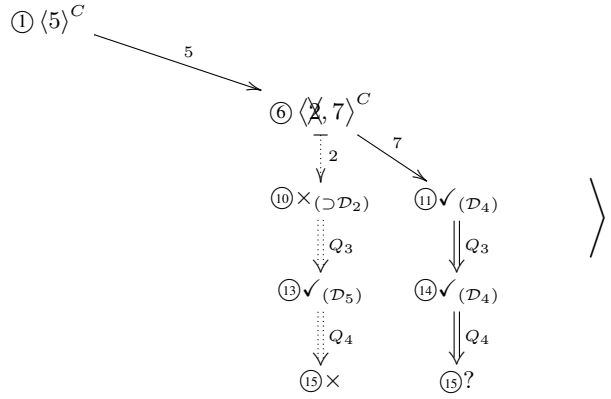
Updated Tree

Iteration 5

$\mathbf{D}_{calc} = \{\mathcal{D}_4\} = \{[5, 7]\}$

$\mathbf{Q} = []$

$\mathbf{C}_{calc} = \{\langle 5\rangle, \langle 7\rangle\}$

$\mathbf{D}_\supset = \emptyset$

$\mathbf{Q}_{dup} = []$

$p_{\mathbf{D}}(\mathcal{D}_4) = 1$

$\Rightarrow \quad$ return the solution KB $(\mathcal{K} \setminus \mathcal{D}_4) \cup Q_4 \qquad \square$
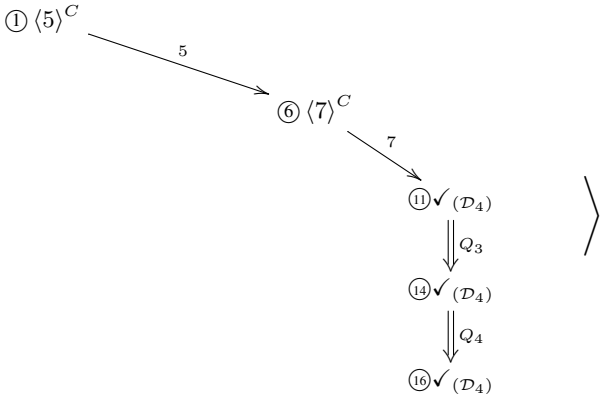
Figure 12.2: (Example 12.1 continued) Solving the problem of Interactive Dynamic KB Debugging (Problem Definition 6.1) for the example DPI given by Table 15.3 by means of Algorithm 5 and DYNAMICHS.

Example 12.2 Let us now consider the (admissible) DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given by Table 4.2. We assume an expert (called user throughout this example) in the domain Dom modeled by $\mathcal{K}$ who wants to find a solution to Interactive Dynamic KB Debugging for the given DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ by means of Algorithm 5 with $mode = dynamic$. Further, the same scenario and parameter settings as in Example 11.2 are supposed. That is, $n_{\min} = n_{\max} = 3$ (notice that the time limit t is irrelevant in this case), $q := 1$ (cf. Chapter 8), $qsm()$ is equal to any query selection measure described in Section 9.3, $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}} : \widetilde{\mathcal{K}} \cup \overline{\mathcal{K}} \to [0, 1]$ is given such that $p_{\mathcal{K}}(ax)$ for $ax \in \mathcal{K}$ resulting from the application of GETAXIOMSPROBS is as given by Table 11.1 and $\sigma := 0$.

The tree constructed and parameters computed and used by Algorithm 5 using DYNAMICHS are visualized by Figures 12.3 and 12.4. We use the same notation as in Figures 4.2, 4.3, 11.1, 11.2, 11.3, 12.1 and 12.2 which is described in Examples 4.8, 4.9, 11.1, 11.2 and 12.1.

After the initialization of variables, Algorithm 5 calls the function GETFORMULAPROBS in line 5 which exploits $p_{\widetilde{\mathcal{K}} \cup \overline{\mathcal{K}}}()$ to calculate the function $p_{\mathcal{K}}()$ giving the fault probabilities of formulas in $\mathcal{K}$ (cf. Sections 4.6.1, 9.2 and Example 4.7).

Then, DYNAMICHS is called for the first time, resulting in the hitting set tree given in the first picture in Figure 12.3. As outlined by the numbers ⓘ indicating at which point in time a node is labeled, the root node (initially the empty set) is labeled first by $\mathcal{C}_1 := \langle 1, 2, 5 \rangle$ and three successor nodes, namely $\mathsf{nd}_1 := [1]$, $\mathsf{nd}_2 := [2]$ as well as $\mathsf{nd}_3 := [5]$ with $\mathsf{nd}_1.\mathsf{cs} = \mathsf{nd}_2.\mathsf{cs} = \mathsf{nd}_3.\mathsf{cs} = [\langle 1, 2, 5 \rangle]$, are added to the queue of open nodes $\mathbf{Q}$. Contrary to Example 12.1, where the tree was built up in breadth-first order, in this example the formula probabilities $p() := p_{\mathcal{K}}()$ given by Table 11.1 are used to assign a probability $p_{nodes}(\mathsf{n})$ to each path n in the tree starting from the root node (cf. Formula 4.6 and Definition 4.9). In this vein, the node corresponding to the outgoing edge of $\mathcal{C}_1$ labeled by the formula with the largest fault probability among all formulas in $\mathcal{C}_1$ is processed next. That is, the node $[1]$ with $p_{nodes}([1]) = 0.41$ (as opposed to the nodes $[2]$ and $[5]$ with 0.25 each) is labeled next. The DLABEL procedure, after checking whether $[1]$ is a non-minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ (check is negative), computes another minimal conflict set $\mathcal{C}_2 := \langle 2, 4, 6 \rangle$ such that $[1] \cap \mathcal{C}_2 = \emptyset$ ($\mathcal{C}_2$ is not hit by the node $[1]$) to constitute a label for node $[1]$. The successor nodes $[1, 2]$, $[1, 4]$ and $[1, 6]$ of $[1]$ are generated and added to the list $\mathbf{Q}$ in a way that the sorting of $\mathbf{Q}$ in descending order of $p_{nodes}()$ is maintained.

Since $[1, 4]$ (0.28) as well as $[1, 6]$ (0.27) have a larger probability (as per $p_{nodes}()$) than the nodes $[2]$ (0.25) and $[5]$ (0.25), $\mathbf{Q}$ is given by $[[1, 4], [1, 6], [2], [5], [1, 2]]$ when it comes to the processing of the next node. Since DYNAMICHS always treats the first node of $\mathbf{Q}$ next, it identifies the first minimal diagnoses $\mathcal{D}_1 := [1, 4]$ and $\mathcal{D}_2 := [1, 6]$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ at steps ③ and ④, respectively. At step ⑤, when node $[2]$ is processed, a minimal conflict set $\mathcal{C}_3 := \langle 1, 3, 4 \rangle$ is computed and set as a label for $[2]$, giving rise to the generation of three further nodes $[2, 1]$, $[2, 3]$ and $[2, 4]$, all with $\mathsf{nd}_i.\mathsf{cs} = [\langle 1, 2, 5 \rangle, \langle 1, 3, 4 \rangle]$.

However, notice that not all of these new nodes are added to $\mathbf{Q}$, contrary to STATICHS (cf. Example 11.2). For, there is already a node $[1, 2]$ corresponding to the set $\{1, 2\}$ in $\mathbf{Q}$. Due to the test performed in line 20, this duplicate node $[2, 1]$ is assigned to the list $\mathbf{Q}_{dup}$ which is expressed in the figure by dup. Since diagnoses are sets, not lists, $[1, 2, ax_1, \ldots, ax_k]$ and $[2, 1, ax_1, \ldots, ax_k]$ constitute one and the same diagnosis and it is irrelevant whether the one or the other is found. Hence, the nodes $[1, 2]$ and $[2, 1]$ are regarded as duplicates. Nevertheless, $\mathsf{nd}_i := [2, 1]$ (with $\mathsf{nd}_i.\mathsf{cs} = [\langle 1, 2, 5 \rangle, \langle 1, 3, 4 \rangle]$) must not be completely deleted as it might be the case that (some successor node of) $\mathsf{nd}_j := [1, 2]$ (with $\mathsf{nd}_j.\mathsf{cs} = [\langle 1, 2, 5 \rangle, \langle 2, 4, 6 \rangle]$) becomes redundant due to the eventual addition of some test case. For example, in case the reason for the redundancy of nd_j is given (only) by a witness of redundancy that is a subset of $\langle 2, 4, 6 \rangle$, nd_j is pruned and replaced by the node nd_i which is still non-redundant.

Thence, only $[2, 3]$ and $[2, 4]$ are added to $\mathbf{Q}$ as successor nodes of the processed node $[2]$. Next, the minimal conflict set $\mathcal{C}_2 = \langle 2, 4, 6 \rangle$ is reused (lines 30-40 in DLABEL) as a label for node $[5]$ with $p_{nodes}([5]) = 0.25$ and the three new nodes $[5, 2]$, $[5, 4]$ as well as $[5, 6]$ are generated and assigned to $\mathbf{Q}$ at step ⑦. Then, the fourth minimal conflict set $\mathcal{C}_4 := \langle 1, 5, 6, 8 \rangle$ is computed to label the node $[2, 4]$ with $p_{nodes}([2, 4]) = 0.18$ and the four new nodes $[2, 4, 1]$, $[2, 4, 5]$, $[2, 4, 6]$ as well as $[2, 4, 8]$ are generated

and assigned to **Q** st step ⑧. At step ⑨, the third minimal diagnosis $\mathcal{D}_3 := [5,4]$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is eventually found and added to $\mathbf{D}_{calc}$ which now has reached a cardinality of $3 = n_{\min} = n_{\max}$ wherefore DYNAMICHS stops and returns i.a. the set of leading diagnoses $\mathbf{D}_{calc} = \{[1,4], [1,6], [5,4]\}$. The returned values are given in the lefthand column in Figure 12.3.

As in Example 11.2, where a debugging session for the same DPI using STATICHS is presented, the first query Q_1 is computed as $\{B \sqsubseteq K\}$ and answered by *true* by the user. The assignment of Q_1 to the positive test cases of the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ brings the opportunity to perform some significant pruning actions (within the function UPDATETREE called at the beginning of the second call of DYNAMICHS). These are shown in the tree with the caption 'Updated Tree' and in the righthand column in Figure 12.3.

As a first step within UPDATETREE, a redundancy check is performed for each diagnosis in $\mathbf{D}_\times$. In this case $\mathbf{D}_\times = \{\mathcal{D}_3\} = \{[5,4]\}$ since $\mathcal{D}_3$ is the only minimal diagnosis that has been ruled out by the most recently added positive test case Q_1. The purpose of the redundancy check is to figure out whether $\mathcal{D}_3$ is redundant w.r.t. the current DPI and must be pruned or whether it might be extended to become a minimal diagnosis w.r.t. the current DPI.

First, the Quick Redundancy Check (QRC) $\text{QX}(\langle \{1,2,6\}, \mathcal{B}, P \cup \{Q_1\}, N \rangle) = \langle 1 \rangle$ (line 50 in DY-NAMICHS) is executed for $\mathcal{D}_3$ where the KB $\{1,2,6\}$ used in this call of QX is obtained by deletion of node $:= \mathcal{D}_3$ from the union of all conflict sets (the elements of node.cs) along the path that corresponds to $\mathcal{D}_3$, i.e. $\{1,2,6\} = (\langle 1,2,5 \rangle \cup \langle 2,4,6 \rangle) \setminus [5,4]$. By means of the QRC it is figured out (line 52 in DY-NAMICHS) that $\mathcal{D}_3$ (and possibly some further nodes) is redundant and can be pruned. This holds since the minimal conflict set $\langle 1,2,5 \rangle$ w.r.t. the last-but-one DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is not a minimal conflict set w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q_1\}, N \rangle_R$ because $\langle 1 \rangle$ returned by QX is already a minimal conflict set w.r.t. the current DPI (cf. Proposition 4.9). We call this minimal conflict set $\langle 1 \rangle$ a *witness of redundancy* for $\mathcal{D}_3$. Hence, all branches in the hitting set tree starting from an outgoing edge of $\langle 1,2,5 \rangle$ labeled by 2 or by 5 can be safely deleted from all collections storing nodes in DYNAMICHS.

An illustration why $\langle 1 \rangle$ "replaces" $\langle 1,2,5 \rangle$ as a minimal conflict set w.r.t. the current DPI can be given as follows: First, $\langle 1,2,5 \rangle$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ as it is a set-minimal subset of $\mathcal{K}$ that entails $\{A \sqsubseteq K\} = n_1 \in N$ and there is no proper subset $\mathcal{C}'$ of $\langle 1,2,5 \rangle$ where $\mathcal{C}' \cup \mathcal{B} \cup U_P$ violates any $r \in R$ or entails any $n \in N$ (see example 4.3 for a detailed explanation). Second, considering the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q_1\}, N \rangle_R$, we have that $\langle 1,2,5 \rangle \cup \mathcal{B} \cup U_{P \cup \{Q_1\}} \models n_1$, too. However, $\{2,5\} = \{B \sqsubseteq G, G \sqsubseteq K\} \models \{B \sqsubseteq K\} = Q_1$ implies that $\mathcal{B} \cup U_{P \cup \{Q_1\}} \supseteq Q_1$ can replace the subset $\{2,5\}$ of the conflict set $\langle 1,2,5 \rangle$. For, formula 1 ($A \sqsubseteq B$) along with Q_1 ($B \sqsubseteq K$) already entails n_1. Further, $\mathcal{B} \cup U_{P \cup \{Q_1\}}$ cannot violate any negative test case $n_i \in N$ or requirement $r_j \in R$ by the admissibility of the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, the fact that Q_1 is a query, Corollary 7.3, Definition 3.6 and Proposition 3.4. Thus, by Definition 4.1, $\langle 1 \rangle$ is in fact a minimal conflict set w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q_1\}, N \rangle_R$.

Now, the first nice thing at this point is that $\langle 1 \rangle$ is not only a witness of redundancy of nodes nd where $\langle 1,2,5 \rangle \in$ nd.cs, but of each nd (in the tree or in the set $\mathbf{Q}_{dup}$ of duplicate nodes) where nd.cs contains a conflict set that is a proper superset of $\langle 1 \rangle$. That is, $\langle 1 \rangle$ also replaces $\langle 1,3,4 \rangle$ as well as $\langle 1,5,6,8 \rangle$. This implicates that two outgoing edges (those labeled by 2 or 5) of $\langle 1,2,5 \rangle$, two outgoing edges (those labeled by 3 or 4) of $\langle 1,3,4 \rangle$ and three outgoing edges (those labeled by 5, 6 or 8) of $\langle 1,5,6,8 \rangle$ can be pruned.

The second nice thing that has an even more significant bearing on tree pruning than the first thing is that $\langle 1 \rangle$ is a witness of redundancy of the conflict set that labels the root node. That is, pruning can take place at the very top of the tree and two of three subtrees rooted at successor nodes of the root node can be pruned. That is, for instance, *within* the rightmost subtree of the root node in the picture with caption 'Updated Tree' in Figure 12.3 no pruning is possible at all since the conflict set $\langle 2,4,6 \rangle$ labels the root node of this subtree and $\langle 1 \rangle$ is not a subset of $\langle 2,4,6 \rangle$. However, this subtree is still redundant since it is connected with the root node by a "redundant" edge labeled by 5. As a consequence, we can observe the pruning of a total of 9 nodes (of altogether 12 nodes in the tree) in only one execution of UPDATETREE.

Now, to receive an impression of the power of tree pruning in DYNAMICHS, the reader is invited to compare the trees used in iterations 2 and 3 in the current example (the bottom left pictures in Figure 12.3 and Figure 12.4) with the trees used in iterations 2 and 3 in Example 11.2 (the bottom picture in Figure 11.2 and the picture in Figure 11.3) which deals with the debugging of the same DPI (just by means of STATICHS instead of DYNAMICHS), uses the same sets of leading diagnoses in each iteration, thus the same queries, and of course the same user (that gives the same answers in both examples).

After all diagnoses of $\mathbf{D}_\checkmark$ are added to $\mathbf{Q}$ as a final action within UPDATETREE, the repeat-loop of the second iteration of DYNAMICHS is entered. Here, the minimal diagnoses $\mathcal{D}_1$ ($p_{nodes}(\mathcal{D}_1) = 0.28$, step ⑪), $\mathcal{D}_2$ (0.27, ⑫) and $\mathcal{D}_4$ (0.09, ⑬) are found and assigned to the empty set $\mathbf{D}_{calc}$ before DYNAMICHS terminates again. Notice that only one call of the DLABEL procedure is required in the second iteration (for node $[1, 2]$) due to the test in line 8 of DYNAMICHS which is positive for $\mathcal{D}_1$ and $\mathcal{D}_2$ (since $\mathcal{D}_1, \mathcal{D}_2 \in \mathbf{D}_\checkmark$).

Once the second query $Q_2 = \{B \sqsubseteq \exists r.F\}$ is added to the positive test cases resulting in the DPI $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q_1, Q_2\}, N \rangle_R$, the UPDATETREE function causes the pruning of two further nodes ($\mathcal{D}_2 = [1, 6]$ and $\mathcal{D}_4 = [1, 2]$) leading to the continuance of only a single node ($\mathcal{D}_1 = [1, 4]$) in the memory of DYNAMICHS (see the picture with caption 'Updated Tree' in Figure 12.4). The reason for this is that Q_2 can "replace" the part $\{2, 6\} = \{B \sqsubseteq G, G \sqsubseteq \exists r.F\}$ (which entails Q_2) of the minimal conflict set $\langle 2, 4, 6 \rangle$ w.r.t. the last-but-one DPI $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q_1\}, N \rangle_R$ such that $\langle 2, 4, 6 \rangle \setminus \{2, 6\} = \langle 4 \rangle$ is already a minimal conflict set w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q_1, Q_2\}, N \rangle_R$ (cf. the analysis of the minimal conflict set $\mathcal{C}_2 = \langle 2, 4, 6 \rangle$ in Example 4.3).

Since, by now, all minimal conflict sets $\langle 1, 2, 5 \rangle$, $\langle 2, 4, 6 \rangle$, $\langle 1, 5, 6, 8 \rangle$ as well as $\langle 1, 3, 4 \rangle$ w.r.t. the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ have "shrunk" as much as to constitute only two different set-minimal sets $\langle 1 \rangle$ and $\langle 4 \rangle$, it is clear by Proposition 4.6 that there can be only a single minimal diagnosis $[1, 4]$ w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q_1, Q_2\}, N \rangle_R$. Therefore, the third iteration of DYNAMICHS terminates due to $\mathbf{Q} = []$ and returns the singleton set $\mathbf{D}_{calc} = \{[1, 4]\}$. Consequently, the probability $p_\mathbf{D}([1, 4]) = 1$ wherefore Algorithm 5 also stops executing and returns $(\mathcal{K} \setminus [1, 4]) \cup p_1 \cup Q_1 \cup Q_2$ as the (exact) solution to the Interactive Dynamic KB Debugging problem for the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

The advantage of DYNAMICHS in this example over STATICHS in Example 11.2 in iterations 2 and 3 is that the pruning of nodes lets the algorithm automatically focus on the still relevant (i.e. non-redundant) parts of the tree. STATICHS, on the other hand, is doomed to spend most of the execution time for investigating nodes that turn out to be already invalidated by some specified test case(s). As already mentioned in Example 11.2, the inability of STATICHS to "early-prune" incomplete branches of the tree is especially unfavorable in the last iteration of STATICHS in case $\sigma = 0$ since all irrelevant minimal diagnoses w.r.t. the input DPI must first be computed before they can be ruled out.

This immense upside of DYNAMICHS over STATICHS (see the analysis in the end of Example 11.2) also finds expression in the quantitative analysis of this example given next. All in all, the execution of Algorithm 5 in this example performs

- 4 full QX calls, i.e. calls of QX using the KB $\mathcal{K} \setminus node$ for a node *node* that actually return a minimal conflict set (there are four minimal conflict sets labeled by C in Figures 12.3 and 12.4 which do not result from QRC, CRC or the minimality test of a conflict set in line 32 of DYNAMICHS),

- 2 fast QX calls, i.e. executions of QX within the scope of the QRC (one call of QX each for the QRC of $\mathcal{D}_3$ and $\mathcal{D}_2$),

- 4 validity checks, i.e. calls of QX that return 'no conflict' (one check for each of the four found minimal diagnoses where the identification of diagnoses $\mathcal{D}_1$ at step ⑪, $\mathcal{D}_2$ at step ⑫ and $\mathcal{D}_1$ at step ⑮ does not require any call to a reasoning service by means of $\mathbf{D}_\checkmark$, see line 8 in DYNAMICHS; notice that QX does only perform a single KB validity check by ISKBVALID in case it returns 'no conflict', see Algorithm 1) and

- 2 tree update processes involving 11 pruned nodes (9 nodes during the first update between steps ⑩ and ⑪ and 2 nodes during the second between steps ⑭ and ⑮),

computes

- 4 minimal diagnoses ($\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$ and $\mathcal{D}_4$, all w.r.t. the input DPI),

- 6 minimal conflict sets ($\langle 1, 2, 5 \rangle$, $\langle 2, 4, 6 \rangle$, $\langle 1, 3, 4 \rangle$ and $\langle 1, 5, 6, 8 \rangle$ w.r.t. the input DPI and the subsets thereof $\langle 1 \rangle$ and $\langle 4 \rangle$ w.r.t. some DPI resulting from the input DPI by addition of new test cases) and

- 2 queries and asks the user 2 logical formulas (1 per query)

and stores

- a maximum of 12 nodes (where node refers to the internal representation of a node nd in DY-NAMICHS as a list of edge labels (nd) and a list of node labels (nd.cs) along a path from the root node to a leaf node).

Finally, we want to emphasize that, in all executions of UPDATETREE throughout this example, the usually very efficient QRC was successful right off and the usually more time-consuming CRC was never required. □

Iteration 1
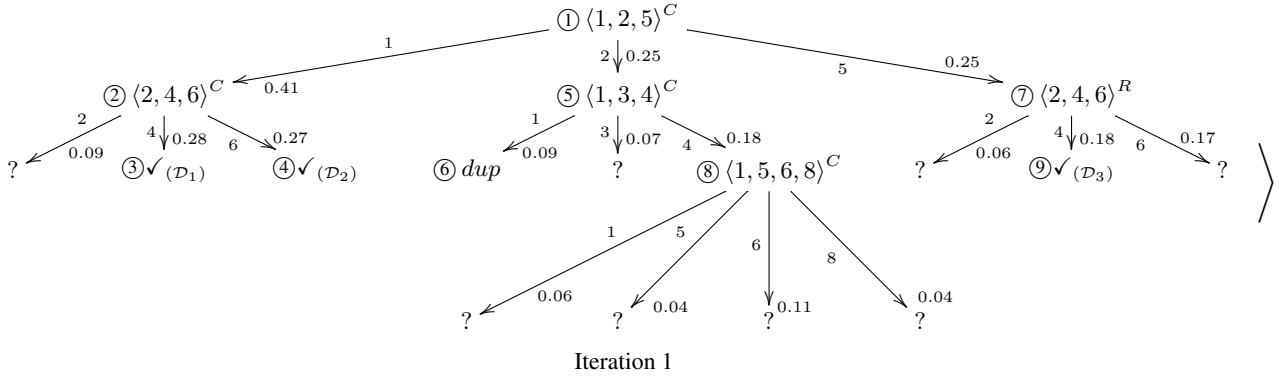
$\mathbf{D}_{calc} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\} = \{[1,4],[1,6],[5,4]\}$

$\mathbf{Q} = [[5,6],[2,4,6],[1,2],[2,3],[5,2],$
$\qquad [2,4,1],[2,4,5],[2,4,8]]$

$\mathbf{C}_{calc} = \{\langle 1,2,5\rangle, \langle 2,4,6\rangle, \langle 1,3,4\rangle, \langle 1,5,6,8\rangle\}$

$\mathbf{D}_{\supset} = \emptyset$

$\mathbf{Q}_{dup} = [[2,1]]$

$\langle Q_1, \mathfrak{P}(Q_1)\rangle = \langle \{B \sqsubseteq K\}, \langle \{\mathcal{D}_1,\mathcal{D}_2\}, \{\mathcal{D}_3\}, \emptyset\rangle\rangle$

$u(Q_1) = true$

$\mathbf{D}_{\checkmark} = \{\mathcal{D}_1, \mathcal{D}_2\}, \mathbf{D}_{\times} = \{\mathcal{D}_3\}$

UPDATETREE:

QRC $(\mathcal{D}_3)$: QX$(\langle \{1,2,6\}, \mathcal{B}, P \cup \{Q_1\}, N\rangle) = \langle 1\rangle$

$\Rightarrow$ PRUNEQDUP/PRUNE: $\langle 1,2,5\rangle \to \langle 1\rangle, \langle 1,3,4\rangle \to \langle 1\rangle$

- prune all subtrees starting from nodes $\langle 1,2,5\rangle$ by an outgoing edge with label 2 or 5
- prune all subtrees starting from nodes $\langle 1,3,4\rangle$ by an outgoing edge with label 3 or 4
- replace by $\langle 1\rangle$ all node labels in the tree that are proper supersets of $\langle 1\rangle$

$\Rightarrow \mathbf{D}_{\supset} = \emptyset, \mathbf{C}_{calc} = \{\langle 1\rangle, \langle 2,4,6\rangle\},$
$\qquad \mathbf{Q} = [[1,4],[1,6],[1,2]], \mathbf{Q}_{dup} = []$

Updated Tree

Iteration 2
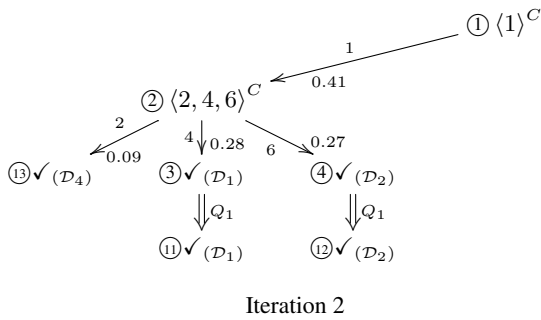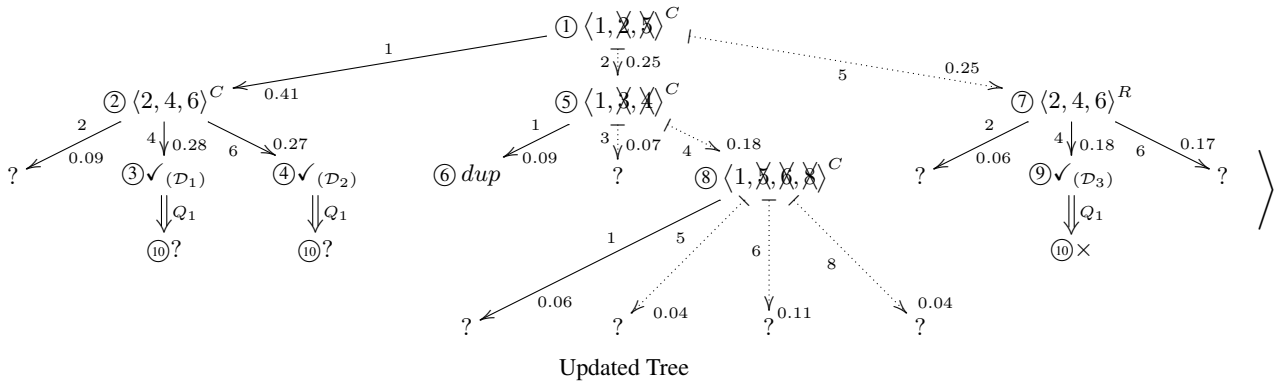
$\mathbf{D}_{calc} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_4\} = \{[1,4],[1,6],[1,2]\}$

$\mathbf{Q} = []$

$\mathbf{C}_{calc} = \{\langle 1\rangle, \langle 2,4,6\rangle\}$

$\mathbf{D}_{\supset} = \emptyset$

$\mathbf{Q}_{dup} = []$

$\langle Q_2, \mathfrak{P}(Q_2)\rangle = \langle \{B \sqsubseteq \exists r.F\}, \langle \{\mathcal{D}_1\}, \{\mathcal{D}_2, \mathcal{D}_4\}, \emptyset\rangle\rangle$

$u(Q_2) = true$

$\mathbf{D}_{\checkmark} = \{\mathcal{D}_1\}, \mathbf{D}_{\times} = \{\mathcal{D}_2, \mathcal{D}_4\}$

Figure 12.3: (Example 12.2) Solving the problem of Interactive Dynamic KB Debugging (Problem Definition 6.1) for the example DPI given by Table 4.2 by means of Algorithm 5 and DYNAMICHS.

UPDATETREE:

QRC $(\mathcal{D}_2)$: $\text{QX}(\langle\{2,4\},\mathcal{B},P\cup\{Q_1,Q_2\},N\rangle)=\langle 4\rangle$

$\Rightarrow$ PRUNE: $\langle 2,4,6\rangle \to \langle 4\rangle$

$\Rightarrow$ $\mathbf{D}_{\supset}=\emptyset$, $\mathbf{C}_{calc}=\{\langle 1\rangle,\langle 4\rangle\}$

$\mathbf{Q}=[[1,4]]$, $\mathbf{Q}_{dup}=[]$

Updated Tree
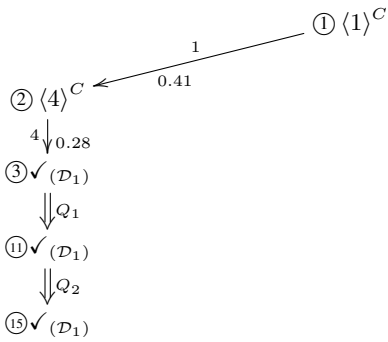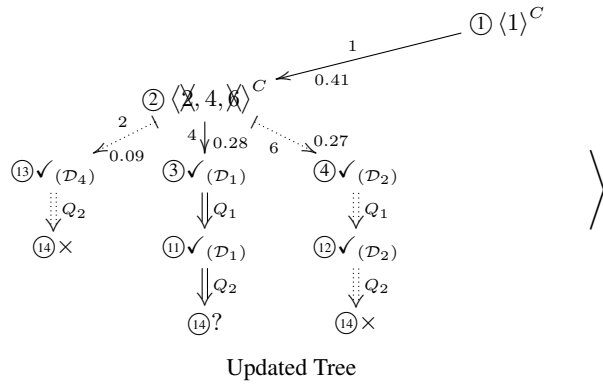
Iteration 3

$\mathbf{D}_{calc}=\{\mathcal{D}_1\}=\{[1,4]\}$

$\mathbf{Q}=[]$

$\mathbf{C}_{calc}=\{\langle 1\rangle,\langle 4\rangle\}$

$\mathbf{D}_{\supset}=\emptyset$

$\mathbf{Q}_{dup}=[]$

$p_{\mathbf{D}}(\mathcal{D}_1)=1$

$\Rightarrow$ return the solution KB $(\mathcal{K}\setminus\mathcal{D}_1)\cup p_1\cup Q_1\cup Q_2$

$(p_1$: cf. Table 4.2$)$ $\quad\square$

Figure 12.4: (Example 12.2 continued) Solving the problem of Interactive Dynamic KB Debugging (Problem Definition 6.1) for the example DPI given by Table 4.2 by means of Algorithm 5 and DYNAMICHS.

12.4 Algorithm Details and Correctness

In this section we will discuss DYNAMICHS in a detailed way and give proofs of its completeness and soundness. To this end, we first give some definitions and some hints regarding the notation used in this section.

12.4.1 Definitions and Notation

The DYNAMICHS algorithm will require a different storage of nodes than STATICHS and Algorithm 2 since it will not interpret different branches with the same set of edge labels in the hitting set tree to be equivalent. So, DYNAMICHS, as opposed to STATICHS and Algorithm 2, will not discard any branch that is a duplicate branch in terms of its edge labels. Instead, a set storing these duplicate branches will be consulted each time a branch is found to be "redundant" and thus needs to be pruned. This strategy enables the substitution of a "redundant" branch by a "non-redundant" branch featuring an equal set of edge labels.

That is why a node nd in (the hitting set tree produced by) DYNAMICHS corresponds to the *ordered* list of edge labels visited when traversing a path from the root node to some leaf node. As an attribute of nd, nd.cs corresponds to the *ordered* list of node labels visited when traversing a path from the root node to some leaf node.

Definition 12.1. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be the DPI and P' and N' the sets of positively and negatively answered queries given as an input to DYNAMICHS. Let further $P_1'', \ldots, P_k''$ and $N_1'', \ldots, N_k''$ such that $P_j'' \subseteq P'$ and $N_j'' \subseteq N'$ for $j \in \{1, \ldots, k\}$. Then we define in DYNAMICHS*

- *a node $\mathsf{nd} = [ax_1, \ldots, ax_k]$ to be an (ordered) list of elements $ax_j \in \mathcal{K}$*

where each node nd stores as an attribute

- *the (ordered) list $\mathsf{nd.cs} = [\mathcal{C}_1, \ldots, \mathcal{C}_k]$ such that $\mathcal{C}_j$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P_j'', N \cup N_j'' \rangle_R$ and $ax_j \in \mathcal{C}_j$ for all $j \in \{1, \ldots, k\}$ corresponding to the set of node labels on the path from the root node to nd.*

Further, $\mathsf{nd}[i]$ refers to the i-th element in nd, i.e. to ax_i, and $\mathsf{nd.cs}[i]$ refers to the i-th element in nd.cs, i.e. to $\mathcal{C}_i$. Notice that conflict sets $\mathsf{nd.cs}[i]$ itself are (non-ordered) sets.
Moreover, we define

- $|\mathsf{nd}|$ *and* $|\mathsf{nd.cs}|$ *to denote the number of elements in the lists nd and nd.cs,*

- $\mathsf{nd}[i..k] := [\mathsf{nd}[i], \ldots, \mathsf{nd}[k]]$ *for $i \leq k$ and $|\mathsf{nd}| \geq k$,*

- $\mathsf{nd.cs}[i..k] := [\mathsf{nd.cs}[i], \ldots, \mathsf{nd.cs}[k]]$ *for $i \leq k$ and $|\mathsf{nd.cs}| \geq k$,*

- *nodes nd and $\mathsf{nd}[i..k]$ appearing on the left or right side of expressions using the following set operators to be considered as (non-ordered) sets: $\supset, \supseteq, \subset, \subseteq, =, \setminus$*

We call

- $\mathsf{nd}[1..k]$ *where ($k < |\mathsf{nd}|$) $k \leq |\mathsf{nd}|$ a (proper) subnode of nd and*

- nd'' *a successor (node) of nd' iff nd' is a proper subnode of nd''.*

- nd the same node as nd' *iff*

  - $|\mathsf{nd}| = |\mathsf{nd}'|$ *and*

– $\mathsf{nd}[i] = \mathsf{nd}'[i]$ *for* $i \in \{1, \ldots, |\mathsf{nd}|\}$ *and*

– $\mathsf{nd.cs}[i] = \mathsf{nd}'.\mathsf{cs}[i]$ *for* $i \in \{1, \ldots, |\mathsf{nd}|\}$.

Example 12.3 For instance, in line 20 of Algorithm 8, the test $\mathsf{node}_e \in \mathbf{Q}$ checks whether there is some set nd in $\mathbf{Q}$ such that node_e and nd interpreted as sets are equal. That is, $\mathsf{node}_e := \{1, 3, 2\}$ is equal to $\mathsf{nd} := \{2, 1, 3\}$ although the order of formulas is different and the ordered sets of conflict sets $\mathsf{node}_e.\mathsf{cs}$ and $\mathsf{nd.cs}$ might be different as well. Another example of this interpretation of nodes as sets can be found in line 50 where $U_{\mathsf{nd.cs}} \setminus \mathsf{nd}$ refers to the set difference of the union of all sets in $\mathsf{nd.cs}$ and the set nd. If, e.g. $U_{\mathsf{nd.cs}} := \{1, 2, 3, 4\}$ and $\mathsf{nd} := \{4, 2\}$, the result of this set difference is $\{1, 3\}$ or, equivalently, $\{3, 1\}$.

On the other hand, if the operator is not one of those listed above, then node is interpreted as an ordered set. For example, consider line 19 where the ADD operator is used to append a logical formula e to the end of the ordered set of formulas node. Suppose, e.g. $\mathsf{node} := [3, 1, 2]$ and $e := 4$, then the result is $[3, 1, 2, 4]$ which is not equal to $[1, 2, 3, 4]$. □

The following definition characterizes alternative paths in a hitting set tree produced by DYNAMICHS, i.e. different paths leading to the same (leaf) node in the tree.

Definition 12.2. *Let* nd *and* nd′ *be nodes in* DYNAMICHS *such that*

- $|\mathsf{nd}'| \leq |\mathsf{nd}|$,

- $\mathsf{nd}' = \mathsf{nd}[1..|\mathsf{nd}'|]$ *and*

- *there is some* $j \in \{1, \ldots, |\mathsf{nd}'|\}$ *with the property that* $\mathsf{nd}'[j] \neq \mathsf{nd}[j]$ *or* $\mathsf{nd}'.\mathsf{cs}[j] \neq \mathsf{nd.cs}[j]$.

Further, let ADD(L_1, L_2) *be the function that outputs the list* $[a_1, \ldots, a_n, b_1, \ldots, b_m]$ *given two lists* $L_1 := [a_1, \ldots, a_n]$ *and* $L_2 := [b_1, \ldots, b_m]$.
Then we call

- nd′ *an* alternative subnode of nd,

- nd′ *a* proper alternative subnode of nd *if* $|\mathsf{nd}'| < |\mathsf{nd}|$ *and*

- node *where*

    – $\mathsf{node} := \mathrm{ADD}(\mathsf{nd}', \mathsf{nd}[|\mathsf{nd}'| + 1..|\mathsf{nd}|])$ *and*

    – $\mathsf{node.cs} := \mathrm{ADD}(\mathsf{nd}'.\mathsf{cs}, \mathsf{nd.cs}[|\mathsf{nd}'.\mathsf{cs}| + 1..|\mathsf{nd.cs}|])$

    an alternative equal node of nd.

- *In a context where* nd′ *is relevant, we call* node *the* alternative equal node of nd constructed from nd′.

Regarded as a set, an alternative equal node node of some node nd is equal to nd. There is just at least one difference between node and nd with regard to the order of elements in nd as opposed to the order of elements in node or with regard to the (order of) elements in nd.cs as opposed to the (order of) elements in node.cs.

Example 12.4 Let $\mathsf{nd} := [1, 2, 3, 4]$ with $\mathsf{nd.cs} := [\langle 1, 2, 3 \rangle, \langle 2, 6 \rangle, \langle 3, 6, 7 \rangle, \langle 4, 5 \rangle]$. Then, $\mathsf{nd}_1 := [2, 1]$ with $\mathsf{nd}_1.\mathsf{cs} := [\langle 1, 2, 3 \rangle, \langle 1, 4 \rangle]$ as well as $\mathsf{nd}_2 := [3, 2, 1]$ with $\mathsf{nd}_2.\mathsf{cs} := [\langle 1, 2, 3 \rangle, \langle 2, 6 \rangle, \langle 1, 4 \rangle]$ are alternative subnodes of nd. To see that nd_1 is an alternative subnode of nd, observe that the set-equality between $\mathsf{nd}_1 = [2, 1]$ and $\mathsf{nd}[1..|\mathsf{nd}_1|] = [1, 2]$ holds and $2 = \mathsf{nd}_1[j] \neq \mathsf{nd}[j] = 1$ for $j := 1$

holds. Similarly, for nd_2, we have that the set equality between $[1, 2, 3]$ and $[3, 2, 1]$ holds and the elements on the j-th position for, e.g. $j := 1$, are different, i.e. $1 \neq 3$.

These alternative subnodes of nd can be used to construct the following alternative equal nodes of nd: The one obtained from nd_1 is $node_1 := [2, 1, 3, 4]$ with $node_1.cs := [\langle 1, 2, 3 \rangle, \langle 1, 4 \rangle, \langle 3, 6, 7 \rangle, \langle 4, 5 \rangle]$ and the one obtained from nd_2 is $node_2 := [3, 2, 1, 4]$ with $node_1.cs := [\langle 1, 2, 3 \rangle, \langle 2, 6 \rangle, \langle 1, 4 \rangle, \langle 4, 5 \rangle]$.   □

The following definition introduces the terminology that will be used throughout this section to refer to nodes in DYNAMICHS with certain properties.

Definition 12.3. *In* DYNAMICHS*, a node* nd *with* nd.cs *is called*

- generated *iff it is built in lines 18 and 19,*

- processed *iff lines 6-15 have been executed for* node := nd*,*

- pruned *iff*

    - *it is found to be redundant in line 91 and no node* $nd'' =$ nd *is added to* S' *in line 100 or*
    - *it is found to be redundant in line 112 and no node* $nd'' =$ nd *is added to* Dup_{new} *in line 121*

- replaced *iff it is found to be redundant in line 91 and some node* $nd_{rep} =$ nd *is added to* S' *in line 100*

- combined-replaced *iff it is found to be redundant in line 112 and some node* $nd_{comb,rep} =$ nd *is added to* Dup_{new} *in line 121*

at any point in time during the execution of DYNAMICHS *at any call to* DYNAMICHS *during the execution of Algorithm 5.*

The node nd_{rep} *is referred to as* replacement node (of nd) *and the node* $nd_{comb,rep}$ *is referred to as* combined replacement node (of nd)*.*

12.4.2 The Labeling Function in DYNAMICHS

The following two lemmata provide an analysis of the DLABEL function and characterize the output given by this function independently of when it is called during the execution of Algorithm 5.

The first one analyzes the case where DLABEL returns *valid* or *nonmin* which means that the node for which DLABEL was called is a diagnosis or a non-minimal diagnosis w.r.t. the current DPI, respectively. Further on, it states that only diagnoses w.r.t. the current DPI can be stored in the set $\mathbf{D}_{calc}$ and only diagnoses for whose non-minimality there is evidence in terms of a diagnosis in $\mathbf{D}_{calc}$ can be labeled by *nonmin*.

Lemma 12.1. *Let the* DLABEL *procedure be called at any point in time during the execution of* DYNAMICHS *given i.a. some node* node*, some DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$*, some set of positive test cases* P' *and some set of negative test cases* N' *as argument. Then the following holds:*

(1) If DLABEL *returns* valid*,* node *is a diagnosis w.r.t. the current DPI* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$*.*

(2) During this execution of DYNAMICHS*,* $\mathbf{D}_{calc}$ *comprises only diagnoses w.r.t. the current DPI* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$*.*

(3) If DLABEL *returns* nonmin*,* node *is a non-minimal diagnosis w.r.t. the current DPI* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$*.*

(4) At the time the label nonmin *is returned for* node*, there is some diagnosis* $\mathcal{D}'$ *w.r.t. the current DPI* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ *such that* $\mathcal{D}' \in \mathbf{D}_{calc}$ *and* node $\supset \mathcal{D}'$*.*

Proof. (1): Assume that DLABEL returns *valid* for node. Then, by Proposition 4.9, Remark 4.3, Corollary 3.3, Corollary 7.3 and the fact that the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ used in DYNAMICHS as an input to DLABEL is the same DPI as the admissible one given as an input to Algorithm 5, node must be a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. This proves proposition (1).

(2): This is a direct conclusion from proposition (1) and the facts that nodes labeled by valid are added to the set $\mathbf{D}_{calc}$ in line 13, at the beginning of the execution of DYNAMICHS, $\mathbf{D}_{calc} = \emptyset$ holds (line 3) and $\mathbf{D}_{calc}$ is modified only in line 13 throughout DYNAMICHS.

(3): At the beginning of the execution of DYNAMICHS, $\mathbf{D}_{calc} = \emptyset$ (line 3) and $\mathbf{D}_{calc}$ is modified only in line 13 throughout DYNAMICHS. In line 13, exactly those nodes are added to $\mathbf{D}_{calc}$ for which the DLABEL function returns *valid*. By the correctness of proposition (1), only diagnoses w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ can be added to $\mathbf{D}_{calc}$.

Now, assume DLABEL returns *nonmin* for node. Then, due to the fact that $\mathbf{D}_{calc}$ can only comprise diagnoses w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ and node $\supset \mathcal{D}'$ for some $\mathcal{D}' \in \mathbf{D}_{calc}$ by line 27, node must be a non-minimal diagnosis w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$.

(4): This is a direct consequence of proposition (3). $\square$

The following lemma states that the set $\mathbf{C}_{calc}$ given as an input to DLABEL must include only minimal conflict sets, each w.r.t. the current DPI or some DPI including only a subset of the test cases the current DPI comprises. Moreover, it provides evidence that, in case DLABEL returns a set, this set is a minimal conflict set w.r.t. the current DPI which is not hit by the node given as input to DLABEL.

Lemma 12.2. *Let the DLABEL procedure be called at any point in time during the execution of DYNAMICHS given i.a. some node* node, *a set of sets* $\mathbf{C}_{calc}$, *some DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, *some set of positive test cases* P' *and some set of negative test cases* N' *as argument. Then,*

(1) each element in $\mathbf{C}_{calc}$ *is a minimal conflict set w.r.t. some DPI* $\langle \mathcal{K}, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R$ *where* $P'' \subseteq P'$ *and* $N'' \subseteq N'$ *and*

(2) if DLABEL returns a set L, *then this set* L *is a minimal conflict set w.r.t. the current DPI* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ *and* node $\cap L = \emptyset$.

Proof. (1): At the first call to DYNAMICHS, $\mathbf{C}_{calc} = \emptyset$ is given as an input argument to DYNAMICHS (lines 1 and 10 in Algorithm 5). The only places throughout DYNAMICHS where $\mathbf{C}_{calc}$ is modified are lines 39, 45 and 66. However, modifications to $\mathbf{C}_{calc}$ in lines 39 and 66 can only take place in case there is already some element in $\mathbf{C}_{calc}$. That is, the first element must be added to $\mathbf{C}_{calc}$ in line 45.

In line 45, only minimal conflict sets w.r.t. some DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R$ are added to $\mathbf{C}_{calc}$ where $P'' \subseteq P'$ and $N'' \subseteq N'$ since the call to DLABEL might have taken place during some prior execution of DYNAMICHS during the execution of Algorithm 5. In order to reach line 45, QX called with the DPI $\langle \mathcal{K} \setminus$ node$, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R$ as argument must not return 'no conflict' (line 41). That is, a minimal conflict set $L \neq \emptyset$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R$ is computed in line 41 by Proposition 4.9, Remark 4.3, Corollary 7.3 and the fact that the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ used in DYNAMICHS as an input to DLABEL is the same DPI as the admissible one given as an input to Algorithm 5.

In lines 39 and 66, the following is true: (*) Only minimal conflict sets that are proper subsets of elements already in $\mathbf{C}_{calc}$ can be added to $\mathbf{C}_{calc}$. In the case of line 39, (*) is true due to the following reasons: In order to reach line 39, QX$(\langle \mathcal{C}, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R) = X \neq \mathcal{C}$ must hold for some element $\mathcal{C} \in \mathbf{C}_{calc}$. Since $\mathbf{C}_{calc}$ is never changed in Algorithm 5 between two calls to DYNAMICHS, $\mathbf{C}_{calc}$ comprises only conflict sets w.r.t. the current DPI or previous DPIs (including fewer test cases than the current one). Moreover, a minimal conflict set $\mathcal{C}$ can only shrink after the addition of new test cases to the DPI for which it was computed by Proposition 12.1. Hence, the newly added element X must be a proper subset of the existing element $\mathcal{C}$ in $\mathbf{C}_{calc}$. That X is a minimal conflict set w.r.t. the DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R$ follows from QX$(\langle \mathcal{C}, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R) = X$, Proposition 4.9,

Remark 4.3, Corollary 7.3 and the fact that the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ used in DYNAMICHS as an input to DLABEL is the same DPI as the admissible one given as an input to Algorithm 5.

In the case of line 66, (*) is true due to the following reasons: Due to Lemmata 12.6 and 12.7, $quickPC = true$ or $completePC = true$ can only hold if X is a witness of redundancy of nd. By Definition 12.4, a witness of redundancy is a conflict set w.r.t. the current DPI which is a proper subset of some conflict set that has been used as a label in nd.cs. However, each label in nd.cs must be an element of $\mathbf{C}_{calc}$ due to lines 30, 45 and 19.

(2): That, in case DLABEL returns a set L, it returns a minimal conflict set w.r.t. the current DPI is a consequence from the inference in the proof of proposition (1). We still need to show that $L \cap$ node $= \emptyset$.

If DLABEL returns in line 46, we can derive from the fact that L is the output of the call $\mathrm{QX}(\langle \mathcal{K} \setminus$ node$, \mathcal{B}, P \cup P', N \cup N' \rangle_R)$, Proposition 4.9 and Definition 4.1 that $L \subseteq \mathcal{K} \setminus$ node which implies that $L \cap$ node $= \emptyset$.

If DLABEL returns in line 34 or line 40, then the return can be executed only if the check $\mathcal{C} \cap$ node $= \emptyset$ is true in line 31. By the argumentation in the proof of proposition (1), for the returned set L it must hold that $L \subseteq \mathcal{C}$. Hence, $L \cap$ node $= \emptyset$ is satisfied. $\qquad \square$

As a simple conclusion from Lemma 12.2, we have that the argument X passed to the PRUNE function called within DLABEL is a minimal conflict set w.r.t. the current DPI:

Corollary 12.1. *Assume the execution of some call to* DYNAMICHS *during the execution of Algorithm 5 using the current DPI* DPI. *Anytime* PRUNE *is called within* DLABEL, *the input* X *given to it is a minimal conflict set w.r.t.* DPI.

Proof. Assume the execution of some call to DYNAMICHS during the execution of Algorithm 5 using the current DPI DPI. Then, Lemma 12.2 says that the set X returned in line 40 is a minimal conflict set w.r.t. DPI. Since X is not modified by any of the functions PRUNE and ADDSETDELSUPSETS, we obtain the proposition of this corollary. $\qquad \square$

From this we derive that the input X passed to PRUNEQDUP called within DLABEL must be a minimal conflict set w.r.t. the current DPI:

Corollary 12.2. *Assume the execution of some call to* DYNAMICHS *during the execution of Algorithm 5 using the current DPI* DPI. *Anytime* PRUNEQDUP *is called within* DLABEL, *the input* X *given to it is a minimal conflict set w.r.t.* DPI.

Proof. This corollary is a direct consequence of Corollary 12.1 and the fact that the argument X given to PRUNEQDUP is the same argument X that is given to PRUNEQDUP (none of these functions modifies X). $\qquad \square$

12.4.3 Impact of Answered Queries on Conflict Sets

After one call to DYNAMICHS in Algorithm 5 returns, the set $\mathbf{D}_{calc}$ (called $\mathbf{D}_{\checkmark}$ in Algorithm 5) returned by DYNAMICHS is used as a set of leading diagnoses w.r.t. the current DPI in order to compute a query. After the answered query is incorporated into the DPI, a new call to DYNAMICHS for this new current DPI is made.

As we have learned from Lemmata 12.1 and 12.2, the new call to DYNAMICHS considers only minimal diagnoses and minimal conflict sets w.r.t. the new current DPI. Therefore, the next proposition investigates the impact of the addition of the answered query as a new test case on the set of minimal conflict sets w.r.t. the new current DPI. Concretely, it claims that the transition from a DPI to a new DPI extended by a test case does change the set of minimal conflict sets, that each (minimal) conflict set remains a (not necessarily minimal) conflict set and that minimal conflict sets cannot grow in size.

It is however important to notice that some "new" minimal conflict set might emerge in the course of this DPI-transition which is not in a subset-relationship with any existing minimal conflict set.

Proposition 12.1. *Let* $\mathbf{D}$ *be a set of minimal diagnoses w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *and* $Q \in \mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. *Further, let either* $P' = P \cup \{Q\}$ *or* $N' = N \cup \{Q\}$. *Then it holds that*

(1) $\mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \neq \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R}$,

(2) each conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *is a conflict set w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$,

(3) each minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *is a conflict set w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$,

(4) there are no $\mathcal{C} \in \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ *and* $\mathcal{C}' \in \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R}$ *such that* $\mathcal{C} \subset \mathcal{C}'$,

(5) if there is a subset-relationship between $\mathcal{C} \in \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ *and* $\mathcal{C}' \in \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R}$, *then* $\mathcal{C}' = \mathcal{C}$ *or* $\mathcal{C}' \subset \mathcal{C}$.

Proof. (1): Assume the opposite, namely that $\mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} = \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R}$. Then, by Proposition 4.6, $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} = \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R}$ must be true. This however is a contradiction to Definition 7.1 and the fact that Q is a query.

(2): Let $\mathcal{C}$ be a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Then $\mathcal{C} \cup \mathcal{B} \cup U_P$ violates some $x \in R \cup N$. If $P' = P \cup \{Q\}$ holds, then, by monotonicity of $\mathcal{L}$, $\mathcal{C} \cup \mathcal{B} \cup U_{P \cup \{Q\}}$ violates some $x \in R \cup N$, i.e. $\mathcal{C}$ is a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$. Otherwise, if $N' = N \cup \{Q\}$ is given, then $\mathcal{C} \cup \mathcal{B} \cup U_P$ violates some $x \in R \cup N \subset R \cup N'$, i.e. $\mathcal{C}$ is a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$.

(3): This is a direct consequence of (2), since each minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

(4): Since, by (3), each minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is also a conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$, there cannot be a minimal conflict set $\mathcal{C}'$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ which is a proper superset of a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ as this would imply non-minimality of $\mathcal{C}'$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$.

(5): This proposition is a direct consequence of (4). $\qquad \square$

Given the existence of some non-empty minimal conflict set w.r.t. an admissible DPI DPI, the extension of the test cases of DPI by a query yields a new DPI DPI' for which all minimal conflict sets are non-empty:

Proposition 12.2. *Let* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *and* $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ *be two DPIs such that* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *is admissible and* $P' \supseteq P$ *and* $N' \supseteq N$ *and* $|P' \cup N'| = |P \cup N| + 1$. *Let further* $\mathcal{C} \neq \emptyset$ *be a minimal conflict set w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *and* $Q \in (P' \cup N') \setminus (P \cup N)$ *be a query w.r.t. some* $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ *and* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. *Then, for each minimal conflict set* $\mathcal{C}'$ *w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ *it holds that* $\mathcal{C}' \neq \emptyset$.

Proof. Assume there is some minimal conflict set $\mathcal{C}'$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ such that $\mathcal{C}' = \emptyset$. This implies that there cannot be a minimal conflict set $\mathcal{C}''$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ which is not the empty set because $\mathcal{C}'$ would be a proper subset of $\mathcal{C}''$, which would be a contradiction to the minimality of $\mathcal{C}''$.

Due to Corollary 7.3 and the fact that a query Q w.r.t. some $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is added to $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ in order to obtain $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$, we have that $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ must be admissible.

By Corollary 3.3, $\mathcal{K}$ cannot be valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$ since $\emptyset$ cannot be a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ by Proposition 4.6 and the fact that $\mathcal{C}$ is a non-empty minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. From this we can infer that $\mathcal{K}$ cannot be valid w.r.t. $\langle \cdot, \mathcal{B}, P', N' \rangle_R$ as $P' \supseteq P$ and $N' \supseteq N$.

Now, by Proposition 4.2, there must be some minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ which is not the empty set, contradiction. $\qquad \square$

12.4.4 Impact of Answered Queries on Diagnoses

Next, we analyze what influence answered queries that are added as new test cases to the current DPI have on the (minimal) diagnoses w.r.t. this DPI. The first lemma assures that each DPI constructed during the execution of Algorithm 5 must be admissible as a consequence of the postulated admissibility of the DPI given as an initial input to Algorithm 5.

Lemma 12.3. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be the DPI and P' and N' the sets of positively and negatively answered queries given as an input to* DYNAMICHS. *Then, the DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ is admissible.*

Proof. The admissibility of $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ follows from the fact that $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is the (coercively) admissible input DPI of Algorithm 5, Corollary 7.3 which reveals that admissibility of a DPI is preserved under the addition of a query to the test cases of the DPI and the fact that P' as well as N' are sets of queries. The latter holds because CALCQUERY (Algorithm 5, line 16) computes only queries and the only place where P' and N' are modified is lines 24-26 where only sets returned by CALCQUERY are added to P' and N'. □

The next proposition confirms the restrictive character of test cases. That is, any extension of a current DPI through the addition of a test case cannot lead to a set of (all) diagnoses w.r.t. the new DPI that is a superset of the set of (all) diagnoses w.r.t. the current DPI. We want to point out that this is not necessarily true for the set of *minimal* diagnoses.

Proposition 12.3. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ be two DPIs such that $P' \supseteq P$ and $N' \supseteq N$. Then, each diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ is also a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Proof. Let $\mathcal{D}' \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R}$. Then, by Corollary 3.3 and Definition 3.2, $(\mathcal{K} \setminus \mathcal{D}') \cup \mathcal{B} \cup U_{P'}$ does not violate any $x \in R \cup N'$. Since however formulas, in particular those in $U_{P' \setminus P}$, that are *added* to a KB cannot invalidate any (unwanted) entailments, in particular those in N', and cannot resolve any inconsistencies or incoherencies by the monotonicity of $\mathcal{L}$, we can conclude that $(\mathcal{K} \setminus \mathcal{D}') \cup \mathcal{B} \cup U_P$ does not violate any $x \in R \cup N'$ either. Since $N' \supseteq N$, non-violation of any test case in N' implies non violation of any test case in N also. Consequently, $(\mathcal{K} \setminus \mathcal{D}') \cup \mathcal{B} \cup U_P$ does not violate any $x \in R \cup N$ and entails all $p \in P$ (due to U_P) wherefore $\mathcal{D}' \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ due to Corollary 3.3 and Definition 3.2. □

As a consequence of this, each minimal diagnosis w.r.t. the new DPI is a diagnosis w.r.t. the current DPI, i.e. either a minimal or a non-minimal diagnosis w.r.t. the current DPI.

Corollary 12.3. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ be two DPIs such that $P' \supseteq P$ and $N' \supseteq N$. Then, each minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ is also a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

Proof. Since Proposition 12.3 holds for all diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$, it also holds for all minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ since each minimal diagnosis is a diagnosis. □

Adding a test case to a DPI cannot make minimal diagnoses shrink:

Proposition 12.4. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ be two DPIs such that $P' \supseteq P$ and $N' \supseteq N$ and let $\mathcal{D} \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. Then, for all $\mathcal{D}' \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R}$, it holds that $\mathcal{D}' \not\subset \mathcal{D}$.*

Proof. Let $\mathcal{D} \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ and let $\mathcal{D}' \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R}$ such that $P' \supseteq P, N' \supseteq N$ and suppose $\mathcal{D}' \subset \mathcal{D}$. By Proposition 12.3, $\mathcal{D}'$ must be a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. By $\mathcal{D}' \subset \mathcal{D}$, this is a contradiction to the premise that $\mathcal{D} \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, i.e. that $\mathcal{D}$ is minimal. □

In fact, it even holds that each "new" minimal diagnosis (which is not a minimal diagnosis w.r.t. the current DPI) resulting from the addition of a test case to the current DPI must be a proper superset of some minimal diagnosis w.r.t. the current DPI. In other words, a minimal diagnosis w.r.t. the new DPI is either a minimal diagnosis w.r.t. the current DPI or a proper superset of some minimal diagnosis w.r.t. the current DPI.

Proposition 12.5. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ be two DPIs such that $P' \supseteq P$ and $N' \supseteq N$ and let $\mathcal{D}' \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R}$ and $\mathcal{D}' \notin \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. Then, there is some $\mathcal{D} \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ such that $\mathcal{D} \subset \mathcal{D}'$.*

Proof. By Corollary 12.3, we know that $\mathcal{D}' \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R}$ is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. If $\mathcal{D}'$ is already a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, then the proposition holds. Otherwise, there must be some $\mathcal{D} \subset \mathcal{D}'$ such that $\mathcal{D}$ is a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. $\qquad \square$

Addition of a query to whatever test case set of a DPI DPI implies that the set of all diagnoses w.r.t. the new DPI is a proper subset of all diagnoses w.r.t. DPI:

Corollary 12.4. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ be two DPIs such that*

- $P' \supseteq P$ *and* $N' \supseteq N$,

- $|P'| = |P| + 1$ *or* $|N'| = |N| + 1$, *but not both, and*

- $(P' \cup N') \setminus (P \cup N) = \{Q\}$ *where Q is a query w.r.t. some set $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ and* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Then, $\mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R} \subset \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ holds.

Proof. By Proposition 12.3 we have that $\mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R} \subseteq \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$. Since $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ results from $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ by the addition of the query Q w.r.t. some set $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ to either P or N, we conclude by Definition 7.1 that at least one minimal diagnosis $\mathcal{D}$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ in $\mathbf{D}$ is not a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$. Assume, $\mathcal{D}$ is a non-minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$. In this case, there must be some $\mathcal{D}' \subset \mathcal{D}$ such that $\mathcal{D}'$ is a minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$. This is a contradiction to Proposition 12.4. Consequently, $\mathcal{D} \notin \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R}$. Hence, $\mathcal{D} \in \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R} \setminus \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R}$. By $\mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R} \subseteq \mathbf{aD}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$, the proposition of the corollary follows. $\qquad \square$

12.4.5 Redundant Nodes in DYNAMICHS

The following result constitutes the basis for the definition of a redundant node we give in the next section. It is already stated in [Rei87], but without a proof. It testifies that the set of all minimal hitting sets of a collection F of sets remains steady if elements that are not set-minimal sets in F are deleted from F. By Proposition 4.6, the same must hold for the set of all minimal diagnoses of the collection of all minimal conflict sets w.r.t. some DPI DPI. That is, considering only minimal hitting sets of minimal conflict sets w.r.t. DPI is sufficient for completeness of a hitting set tree algorithm concerning the finding of all minimal diagnoses w.r.t. DPI.

However, we proved by Proposition 12.1 that existing conflict sets will tend to shrink gradually through the specification of new test cases. This implicates that more and more nodes nd_i stored by DYNAMICHS will have the property that $\mathsf{nd}_i.\mathsf{cs}$ will include non-minimal conflict sets w.r.t. the current DPI which constitutes the first of two criteria that are together sufficient for a safe pruning of nd_i. By safe pruning we mean the deletion of a node without eliminating any minimal diagnoses w.r.t. the current DPI.

Proposition 12.6. *If F is a collection of sets, and if $S \in F$ and $S' \in F$ such that $S \subset S'$, then $F_{sub} := F \setminus \{S'\}$ has the same minimal hitting sets as F.*

Proof. Let $\mathcal{D}$ be a minimal hitting set of F_{sub}, then $\mathcal{D}$ is a hitting set of F since $\mathcal{D} \cap S \neq \emptyset$ holds which implies by $S \subset S'$ that $\mathcal{D} \cap S' \neq \emptyset$. Assume that $\mathcal{D}$ is a non-minimal hitting set of F, i.e. that a subset $\mathcal{D}' \subset \mathcal{D}$ is a hitting set of F. Then, however, by minimality of $\mathcal{D}$ w.r.t. F_{sub} we have that not all sets in

F_{sub} are hit by $\mathcal{D}'$ and thus, by $F_{sub} \subset F$, that not all sets in F can be hit by $\mathcal{D}'$, contradiction. Thus, each minimal hitting set of F_{sub} is also a minimal hitting set of F.

Let $\mathcal{D}$ be a minimal hitting set of F, then $\mathcal{D}$ is clearly a hitting set of $F_{sub} \subset F$. Suppose that $\mathcal{D}$ is a non-minimal hitting set of F_{sub}, i.e. that a proper subset of $\mathcal{D}$ is a hitting set of F_{sub}. Let $\mathcal{D}' \subset \mathcal{D}$ be a subset-minimal such subset of $\mathcal{D}$. That is, $\mathcal{D}'$ is a minimal hitting set of F_{sub}. Since $\mathcal{D}$ is a minimal hitting set of F, $\mathcal{D}'$ is not a (minimal) hitting set of F, but a minimal hitting set of F_{sub}. This is a contradiction to the already proven fact that any minimal hitting set of F_{sub} is also a minimal hitting set of F. $\qquad\square$

Assume the first criterion for a safe pruning of a node nd_i, namely the existence of some non-minimal conflict set w.r.t. the current DPI in nd_i.cs, is met. Then, we have not yet any evidence that nd_i is obsolete since for each of the non-minimal conflict sets in nd_i.cs there must be one (or multiple) proper subset(s) which is a minimal conflict set w.r.t. the current DPI. Let $\mathcal{C}_{\neg min}$ be one particular non-minimal conflict set in nd_i.cs and let $\mathcal{C}$ be the particular proper subset of $\mathcal{C}_{\neg min}$ that is the first "witness" found by DYNAMICHS which documents the non-minimality of $\mathcal{C}_{\neg min}$. Then $\mathcal{C}_{\neg min}$ can be split into two disjoint parts, namely $\mathcal{C}$ and the set of formulas $\overline{\mathcal{C}}$ that $\mathcal{C}_{\neg min}$ does not share with $\mathcal{C}$.

Now, the second criterion for a safe pruning of nd_i is about whether nd_i hits $\overline{\mathcal{C}}$. If so, then nd_i is not a (partial) hitting set of *only* minimal conflict sets w.r.t. the current DPI. Put another way, this means that, under the assumption that a wpHS-tree was constructed using only the "static" current DPI, then the label $\mathcal{C}_{\neg min}$ would have never been produced and hence the node nd_i could have never been generated. Eventually, by the considerations made in Sections 4.6.3 and 11.4, we know that such a static hitting set tree algorithm is complete although not taking into account nodes like nd_i.

These thoughts motivate the following definition of a redundant node[28].

Definition 12.4. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be the DPI and P' and N' the sets of positively and negatively answered queries given as an input to* DYNAMICHS. *Further, let* nd *be a node in* DYNAMICHS. *Then we call* nd *a* redundant node *w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ *iff there is*

- *some $r \in \{1, \dots, |\mathsf{nd}|\}$ and*
- *some minimal conflict set $\mathcal{C}$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$*

such that

- $\mathcal{C} \subset \mathsf{nd}.\mathsf{cs}[r]$ *and*
- $\mathsf{nd}[r] \in \mathsf{nd}.\mathsf{cs}[r] \setminus \mathcal{C}$.

Moreover, $\mathcal{C}$ is called a witness of redundancy *of* nd.

A node node in DYNAMICHS can be only redundant w.r.t. a DPI DPI if $\mathsf{node}.\mathsf{cs}$ comprises some non-minimal conflict set w.r.t. DPI:

Corollary 12.5. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be the DPI and P' and N' the sets of positively and negatively answered queries given as an input to* DYNAMICHS. *Further, let* nd *be a node in* DYNAMICHS *such that $\mathsf{nd}[i] \in \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ for all $i \in \{1, \dots, |\mathsf{nd}|\}$. Then* nd *is not a redundant node w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$.

Proof. Since $\mathsf{nd}.\mathsf{cs}$ comprises only minimal conflict sets w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, there cannot be any $\mathcal{C} \in \mathbf{mC}_{\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R}$ such that $\mathcal{C} \subset \mathsf{nd}.\mathsf{cs}[i]$ for some i. $\qquad\square$

A node that is redundant w.r.t. some DPI DPI remains redundant w.r.t. any DPI' that includes a superset of the test cases DPI includes:

---

[28]We adopt the term "redundant" from [Rei87] where is was informally used in the same context.

Lemma 12.4. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ be two DPIs such that $P' \supseteq P$ and $N' \supseteq N$. Further, let* nd *be a redundant node w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. *Then,* nd *is a redundant node w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$.

Proof. By Proposition 12.1, if $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ results from the addition of a single new positive or negative test case to $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, there cannot be any minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$ that is a proper superset of a minimal conflict w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. By Definition 12.4, we can derive that any redundant node w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ must be a redundant node w.r.t. $\langle \mathcal{K}, \mathcal{B}, P', N' \rangle_R$. The proposition of this lemma is a consequence of further applications of Proposition 12.1. $\square$

This implies that a redundant node that is deleted during the execution of DYNAMICHS using the current DPI DPI cannot become non-redundant throughout the entire remaining execution of the interactive debugging session, i.e. the execution of Algorithm 5. Reason for this is that the sets of test cases in a DPI can only be extended and not reduced in the course of debugging.

Remark 12.7 Note that this has consequences on the way how "mind-changes" of a user might be handled by the interactive algorithm. It implies that the current state of DYNAMICHS (stored in the output variables of DYNAMICHS) cannot be exploited in case a user decides to discard some already answered query or to switch the already submitted answer of some query, resulting in some modified DPI DPI'. In such a situation a new construction of a hitting set tree by DYNAMICHS using the DPI DPI' is indicated. Otherwise, some already pruned redundant node w.r.t. DPI might become a relevant node for DPI' which would lead to a violation of the postulated completeness of DYNAMICHS w.r.t. each current DPI, in this case the DPI DPI'. $\square$

The following result is straightforward and claims that each successor node of a redundant node nd_i w.r.t. DPI is a redundant node w.r.t. DPI. So, if r is the minimal value such that both criteria of Definition 12.4 hold for nd_i, all successor nodes of the subnode $\mathsf{nd}_i[1..r]$ of nd_i can be deleted. In other words, the entire subtree (of the hitting set tree produced by DYNAMICHS) rooted at an outgoing edge e of a non-minimal conflict set where e is labeled by an element ax which is not an element of a given witness of redundancy is obsolete.

Lemma 12.5. *Let* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *be a DPI,* nd *be a redundant node w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *and* nd′ *be a successor node of* nd. *Then,* nd′ *is a redundant node w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.

Proof. The proposition of this lemma is a direct consequence of Definition 12.4. $\square$

12.4.6 Hitting Set Tree Pruning in DYNAMICHS

The main pruning operations performed by DYNAMICHS take place in the scope of the UPDATETREE function which is called right at the beginning of the execution of each call to DYNAMICHS. Assume a call to DYNAMICHS during Algorithm 5 given i.a. the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and the test cases P' and N' as arguments and suppose the last-but-one call to DYNAMICHS was given P'' and N'' as arguments. The job of UPDATETREE is to restore the parameters that store the state of DYNAMICHS (for DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R$) in a way that they include at least all nodes that would be included by the respective parameters produced by a call to DYNAMICHS for the static DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$.

Roughly speaking, this involves the following actions:

- *Pruning:* That is, only nodes that are definitely redundant w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ are deleted. A node is definitely redundant if a witness of redundancy of it is known.

- *Replacement:* A deleted redundant node is replaced by an alternative equal node of it which is non-redundant w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$, if there is such a one. Alternative equal nodes are constructed from the list of duplicate nodes $\mathbf{Q}_{dup}$.

- *Rearrangement:* the reassignation of nodes to $\mathbf{Q}$ that "survived" all pruning steps or were introduced in the course of a replacement step and for which no evidence w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ is given that it should be assigned to any other set.

More concretely, UPDATETREE has the following effect on the collections $\mathbf{Q}$, $\mathbf{D}_\times$, $\mathbf{D}_\supset$, $\mathbf{Q}_{dup}$ which are, together with $\mathbf{D}_\checkmark$, the only node-storing collections of DYNAMICHS at the beginning of the execution of each call to DYNAMICHS:

(a) If nd is in $\mathbf{Q}_{dup}$, then nd is removed from $\mathbf{Q}_{dup}$ only if there is a known witness of redundancy of nd w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. If there is an alternative equal replacement node nd$'$ of nd which is constructable from some node in $\mathbf{Q}_{dup}$, then nd$'$ is added to $\mathbf{Q}_{dup}$.

(b) If nd is in $\mathbf{Q}$, then nd is removed from $\mathbf{Q}$ only if there is a known witness of redundancy of nd w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. If there is an alternative equal replacement node nd$'$ of nd which is constructable from some node in $\mathbf{Q}_{dup}$, then nd$'$ is added to $\mathbf{Q}$.

(c) If nd is in $\mathbf{D}_\times$ and there is no known witness of redundancy of nd w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$, then nd is added to $\mathbf{Q}$.

(d) If nd is in $\mathbf{D}_\times$ and nd is redundant w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$, then, if there is some alternative equal replacement node nd$'$ of nd which is constructable from some node in $\mathbf{Q}_{dup}$, then nd$'$ is added to $\mathbf{Q}$.

(e) If nd is in $\mathbf{D}_\supset$, there is no known witness of redundancy of nd w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ and there is no known minimal diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ which is a proper subset of nd, then nd is added to $\mathbf{Q}$.

(f) All nodes nd in $\mathbf{D}_\checkmark$ are added to $\mathbf{Q}$.

Some comments: Step (a) is conducted by PRUNEQDUP before PRUNE is called, for each witness of redundancy X of some node detected during the execution of UPDATETREE. PRUNE is the function that prunes or replaces nodes that are elements of any other collection than $\mathbf{Q}_{dup}$, i.e. $\mathbf{Q}$, $\mathbf{D}_\times$ or $\mathbf{D}_\supset$, and for which X is a witness of redundancy. In this vein, the PRUNE function just needs to perform a test whether there is *any* node in $\mathbf{Q}_{dup}$ that enables the construction of a replacement node of a deleted node. No check for redundancy of nodes in $\mathbf{Q}_{dup}$ is necessary at this stage since $\mathbf{Q}_{dup}$ has already been processed and cleaned from all redundant nodes w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$.

Under the assumption that the deletion of a node redundant w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ is safe in terms of completeness of DYNAMICHS as to finding all minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ (which we will prove throughout this section), UPDATETREE acts safely. That is, deletion actions are performed just on the basis of given evidence in the form of a witness of redundancy. However, it must be accentuated that this does not necessarily imply the pruning or replacement of *all* redundant nodes w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. This is quite desired as guaranteeing complete pruning might be very costly concerning execution time since it would involve the precomputation of all not-yet-computed minimal conflict sets w.r.t. the current DPI *at once*. In the bad case, since these computations would take place online, i.e. between two successive queries shown to the user, this would be anything but beneficial for an interactive algorithm whose usability and usefulness depends greatly on its timeliness. Apart from that, a single newly added test case can be expected to lead to the introduction of only a small number of minimal conflict sets w.r.t. the current DPI that are no minimal conflict sets w.r.t. the last-but-one DPI.

Which nodes are pruned throughout UPDATETREE depends on which witnesses of redundancy are found, i.e. which minimal conflict sets are computed. The UPDATETREE function is implemented to search targeted for witnesses of redundancy of stored nodes. That is, instead of just computing any minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$, it focuses on the set of nodes $\mathbf{D}_\times$ which includes

the subset of all minimal diagnoses $\mathbf{D}_{calc}$ computed in the last-but-one iteration of DYNAMICHS w.r.t. the last-but-one DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R$, which are no diagnoses w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. Note that we will prove later in this section that $\mathbf{D}_{calc}$, and thus $\mathbf{D}_\times$ and $\mathbf{D}_\checkmark$ which are subsets thereof, will indeed comprise only minimal diagnoses. So, UPDATETREE looks for witnesses of redundancy by means of exactly these minimal diagnoses that have been invalidated through the addition of the most recent answered query to the test cases of the DPI. Each diagnosis nd w.r.t. the last-but-one DPI can be invalidated only because it does not hit some minimal conflict set w.r.t. the current DPI and not because it is a non-minimal hitting set of all minimal conflict sets w.r.t. the current DPI. This can be directly inferred from Proposition 12.4 which manifests that minimal diagnoses cannot shrink by the addition of a new test case i.e. there cannot be any minimal diagnosis w.r.t. the current DPI which is a proper subset of nd.

Now, two cases can be identified for a minimal conflict set $\mathcal{C}$ w.r.t. the current DPI that is not hit by nd:

C1: $\mathcal{C}$ is not in a subset-relationship with any minimal conflict set in nd.cs. That is, $\mathcal{C}$ is definitely not a witness of redundancy of nd.

C2: $\mathcal{C}$ is in a subset-relationship with some minimal conflict set in nd.cs. That is, $\mathcal{C}$ satisfies the first criterion of a witness of redundancy of nd (cf. Definition 12.4). Thence, $\mathcal{C}$ might be a witness of redundancy of nd.

Now, the idea is to try to figure out very fast some $\mathcal{C}$ for a node nd $\in \mathbf{D}_\times$ such that $\mathcal{C}$ is a witness of redundancy of nd. This idea is implemented in the so-called *Quick Redundancy Check (QRC)* which

- calls QX just once given the DPI $\langle U_{\mathsf{nd.cs}} \setminus \mathsf{nd}, \mathcal{B}, P, N \rangle_R$ with the usually very small KB $U_{\mathsf{nd.cs}} \setminus \mathsf{nd} \subseteq \mathcal{K}$ in order to calculate just one minimal conflict set $\mathcal{C}$ w.r.t. the current DPI

- and then verifies whether $\mathcal{C}$ is a witness of redundancy of nd by conducting at most $|\mathsf{nd}|$ subset-relationship checks.

The following lemma confirms that QRC (lines 50-54 in Algorithm 9), if successful, indeed computes a witness of redundancy of nd and thus gives evidence that nd is redundant w.r.t. the current DPI.

Lemma 12.6 (Quick Redundancy Check – QRC)**.** *Let* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ *be the DPI and* P' *and* N' *the sets of positively and negatively answered queries given as an input to* DYNAMICHS. *Further, let* nd *be some node in* DYNAMICHS. *Then the following holds:*

If $QX(\langle U_{\mathsf{nd.cs}} \setminus \mathsf{nd}, \mathcal{B}, P \cup P', N \cup N' \rangle_R)$ *returns a set* $\mathcal{C}$ *such that* $\mathcal{C} \subset \mathsf{nd.cs}[i]$ *for some* $i \in \{1, \ldots, |\mathsf{nd.cs}|\}$, *then*

- nd *is a redundant node w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ *and*

- $\mathcal{C}$ *is a witness of redundancy of* nd*.*

Proof. First, $U_{\mathsf{nd.cs}} \setminus \mathsf{nd}$ includes all elements in the union of all conflict sets in nd.cs except for the elements occurring in nd. So, if $QX(\langle U_{\mathsf{nd.cs}} \setminus \mathsf{nd}, \mathcal{B}, P \cup P', N \cup N' \rangle_R)$ returns a set $\mathcal{C}$, then $\mathcal{C}$ is a minimal conflict set w.r.t. $\langle U_{\mathsf{nd.cs}} \setminus \mathsf{nd}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ by Proposition 4.9.

By Definition 4.1, $\mathcal{C} \subseteq U_{\mathsf{nd.cs}} \setminus \mathsf{nd}$ holds wherefore $\mathcal{C} \cap \mathsf{nd} = \emptyset$. By $\mathcal{K} \supseteq U_{\mathsf{nd.cs}} \setminus \mathsf{nd}$ and Remark 4.3, $\mathcal{C}$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$.

If $\mathcal{C} \subset \mathsf{nd.cs}[i]$ for some $i \in \{1, \ldots, |\mathsf{nd.cs}|\}$, then we have that $\mathcal{C}$ is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ which is a proper subset of nd.cs[i]. Since $\mathcal{C} \cap \mathsf{nd} = \emptyset$ implies that $\mathsf{nd}[i] \notin \mathcal{C}$ for all $i \in \{1, \ldots, |\mathsf{nd}|\}$, we conclude that $\mathsf{nd}[i] \in \mathsf{nd.cs}[i] \setminus \mathcal{C}$. Now, by Definition 12.4, nd is a redundant node w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ and $\mathcal{C}$ is a witness of redundancy of nd. $\square$

Remark 12.8 Please notice that the opposite direction does not necessarily hold. That is, if the node nd is redundant w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$, $\mathrm{QX}(\langle U_{\mathsf{nd.cs}} \setminus \mathsf{nd}, \mathcal{B}, P \cup P', N \cup N' \rangle_R)$ might return

- some $\mathcal{C}$ which is not a subset of any conflict set in nd.cs or

- 'no conflict'. □

As an illustration of that remark, we give the following example:

Example 12.5 For instance, assume a node $\mathsf{nd} = [1, 2]$ with $\mathsf{nd.cs} = [\langle 1, 2, 3 \rangle, \langle 2, 4, 5 \rangle]$ and that $\langle 2, 3 \rangle$ is a minimal conflict set w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ wherefore nd is redundant by Definition 12.4. Then $U_{\mathsf{nd.cs}} \setminus \mathsf{nd} = \{3, 4, 5\}$.

Suppose that $\langle 3, 5 \rangle$ is a minimal conflict set w.r.t. the current DPI as well. So, in this case, $\mathrm{QX}(\langle \{3, 4, 5\}, \mathcal{B}, P \cup P', N \cup N' \rangle_R)$ might return $\langle 3, 5 \rangle$. However, $\langle 3, 5 \rangle$ is neither a subset of $\langle 1, 2, 3 \rangle$ nor a subset of $\langle 2, 4, 5 \rangle$ wherefore $\langle 3, 5 \rangle$ is no witness of redundancy of nd.

On the other hand, if we suppose that $\langle 2, 3 \rangle$ and $\langle 2, 4, 5 \rangle$ are the only minimal conflict sets w.r.t. the current DPI that are subsets of $U_{\mathsf{nd.cs}} = \{1, 2, 3, 4, 5\}$, then 'no conflict' is the output of the call to QX. This holds since $\mathsf{nd}[2] = 2$ is an element of both $\langle 2, 3 \rangle$ and $\langle 2, 4, 5 \rangle$ and hence not an element of $U_{\mathsf{nd.cs}} \setminus \mathsf{nd} = \{3, 4, 5\}$. Therefore, neither $\langle 2, 3 \rangle$ nor $\langle 2, 4, 5 \rangle$ is returned by QX since $\mathrm{QX}(\langle \{3, 4, 5\}, \mathcal{B}, P \cup P', N \cup N' \rangle_R)$ can only return a set that is a subset of $\{3, 4, 5\}$ by Proposition 4.9 and Definition 4.1. □

In both cases of the previous example, an existing witness of redundancy of nd is not detected by QRC. In this situation, i.e. when QRC is negative, a *Complete Redundancy Check (CRC)* is performed which involves QX investigating all the DPIs $\langle \mathsf{nd.cs}[i] \setminus \mathsf{nd}[i], \mathcal{B}, P \cup P', N \cup N' \rangle_R$ for $i \in \{1, \ldots, |\mathsf{nd}|\}$ separately. CRC, as substantiated by the following lemma, does find a witness of redundancy if the node nd is redundant w.r.t. the current DPI; and, if CRC does not find a witness of redundancy w.r.t. the current DPI, then nd is non-redundant w.r.t. the current DPI.

Lemma 12.7 (Complete Redundancy Check – CRC). *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be the DPI and P' and N' the sets of positively and negatively answered queries given as an input to* DYNAMICHS. *Further, let* nd *be some node in* DYNAMICHS. *Then, the following holds:*

(1) nd *is redundant w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ iff there is some $i \in \{1, \ldots, |\mathsf{nd}|\}$ such that $\mathrm{QX}(\langle \mathsf{nd.cs}[i] \setminus \{\mathsf{nd}[i]\}, \mathcal{B}, P \cup P', N \cup N' \rangle_R) = X$ where $X \neq$ 'no conflict'.*

(2) If there is some $i \in \{1, \ldots, |\mathsf{nd}|\}$ such that $\mathrm{QX}(\langle \mathsf{nd.cs}[i] \setminus \{\mathsf{nd}[i]\}, \mathcal{B}, P \cup P', N \cup N' \rangle_R) = X$ where $X \neq$ 'no conflict', then X is a witness of redundancy of nd.

Proof. (1): "$\Leftarrow$": Assume there is some $i \in \{1, \ldots, |\mathsf{nd}|\}$ such that $\mathrm{QX}(\langle \mathsf{nd.cs}[i] \setminus \{\mathsf{nd}[i]\}, \mathcal{B}, P \cup P', N \cup N' \rangle_R) = X$ where $X \neq$ 'no conflict'. Then, by Proposition 4.9, we have that X is a minimal conflict set w.r.t. $\langle \mathsf{nd.cs}[i] \setminus \{\mathsf{nd}[i]\}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ such that $X \subseteq \mathsf{nd.cs}[i] \setminus \{\mathsf{nd}[i]\}$. By Definition 4.1, X is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. Hence, we can conclude that $\mathsf{nd}[i] \notin X$. By Definition 12.1 and since nd is a node in DYNAMICHS, it holds that $\mathsf{nd}[i] \in \mathsf{nd.cs}[i]$. As a consequence, $\mathsf{nd}[i] \in \mathsf{nd.cs}[i] \setminus X$ holds. By Definition 12.4, nd is redundant w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ (and X is a witness of redundancy of nd).

"$\Rightarrow$": Suppose nd is a redundant node w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. Then, by Definition 12.4, there must be some $r \in \{1, \ldots, |\mathsf{nd}|\}$ and some minimal conflict set X w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ such that (i) $X \subset \mathsf{nd.cs}[r]$ and (ii) $\mathsf{nd}[r] \in \mathsf{nd.cs}[r] \setminus X$. By (ii), $\mathsf{nd}[r] \notin X$. By Definition 12.1 and the fact that nd is a node in DYNAMICHS, we obtain that $\mathsf{nd}[r] \in \mathsf{nd.cs}[r]$ must be true. Hence, by (i), we derive that $X \subseteq \mathsf{nd.cs}[r] \setminus \{\mathsf{nd}[r]\}$. By Proposition 4.9, QX given some DPI DPI outputs a minimal conflict set w.r.t. DPI iff there is a minimal conflict set w.r.t. DPI. Therefore and since $\mathrm{QX}(\langle \mathsf{nd.cs}[i] \setminus \{\mathsf{nd}[i]\}, \mathcal{B}, P \cup P', N \cup N' \rangle_R)$ is called for each $i \in \{1, \ldots, |\mathsf{nd}|\}$, it must also be called

for $i := r$ since $r \in \{1, \ldots, |\mathsf{nd}|\}$. So, some minimal conflict set X', and not 'no conflict', must be returned by $\mathrm{QX}(\langle \mathsf{nd.cs}[r] \setminus \{\mathsf{nd}[r]\}, \mathcal{B}, P \cup P', N \cup N' \rangle_R)$ since there is at least one minimal conflict set w.r.t. $\langle \mathsf{nd.cs}[r] \setminus \{\mathsf{nd}[r]\}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$, namely X.

(2): This proposition follows directly from (1) (" $\Leftarrow$ "). $\qquad \square$

At the point where some witness of redundancy X of some node $\mathsf{nd} \in \mathbf{D}_\times$ is found by QRC or CRC in UPDATETREE, the next steps (lines 62-65) involve the pruning of $\mathbf{Q}_{dup}$, $\mathbf{Q}$, $\mathbf{D}_\times$ and $\mathbf{D}_\supset$. As already mentioned, $\mathbf{Q}_{dup}$ is the first collection to be cleaned from redundant nodes (w.r.t. the witness X) in PRUNEQDUP in order to constitute an input to the PRUNE function that does not include any redundant nodes (w.r.t. the witness X) and can be used "blindly" to construct replacement nodes of redundant nodes (w.r.t. the witness X) deleted from $\mathbf{Q}$, $\mathbf{D}_\times$ or $\mathbf{D}_\supset$.

Before any pruning steps have ever been executed during the execution of Algorithm 5, $\mathbf{Q}_{dup}$ comprises all generated nodes nd_{dup} for which, at generation time, there was one node $\mathsf{nd} \in \mathbf{Q}$ such that $\mathsf{nd}_{dup} = \mathsf{nd}$. That means, nd_{dup} is stored in $\mathbf{Q}_{dup}$ in order to be available as an alternative equal node of nd or as an alternative subnode of some successor of nd in case nd is found to be redundant w.r.t. some current DPI.

If some node nd_{dup} in $\mathbf{Q}_{dup}$ is found to be redundant w.r.t. the current DPI, there might be other nodes in $\mathbf{Q}_{dup}$ from which a non-redundant alternative equal node nd'_{dup} of nd_{dup} w.r.t. the current DPI can be constructed. By Definition 12.3, we call such a node nd'_{dup} a combined replacement node of nd_{dup}. The name stems from the fact that nd'_{dup} is generated as a combination of existing nodes in $\mathbf{Q}_{dup}$. Combining two nodes $\mathsf{nd}_1, \mathsf{nd}_2 \in \mathbf{Q}_{dup}$ such that nd_1 is a proper alternative subnode of nd_2 yields nd_3 with $\mathsf{nd}_2 = \mathsf{nd}_3$. nd_3 is constructed in that the first (redundant) part of nd_2 (and $\mathsf{nd}_2.\mathsf{cs}$) is replaced by the (non-redundant) part nd_1 (and $\mathsf{nd}_1.\mathsf{cs}$).

Such a combination is "legitimate" since it gives a node nd_3 that would have been constructed if all duplicate nodes would have been added to $\mathbf{Q}$ and processed regularly instead of being added to $\mathbf{Q}_{dup}$. The strategy to store duplicate nodes (where "duplicate" refers to the *set* a node represents) in a separate collection $\mathbf{Q}_{dup}$ as soon as they are found is part of the space-saving policy the DYNAMICHS algorithm pursues. For, in general, this prevents the algorithm to generate and store exponentially many nodes corresponding to equal sets. Since diagnoses are sets and not lists like nodes, it suffices to find *only one* node corresponding to a diagnosis. Only if some active node (one that is not in $\mathbf{Q}_{dup}$) becomes redundant, some other set-equal node, if available, is constructed from the stored duplicate nodes. This idea is very similar to the way pruning is handled in the directed acyclic graph described in [GSW89].

The idea of node combination is formalized by the following definition.

Definition 12.5. *Let S be a collection of nodes in* DYNAMICHS *and let S_i be the set of nodes of cardinality i in S. Further, let the set $Comb_1(S) := S_1$ and let $Comb_i(S)$ comprise*

- *all nodes in S_i and*

- *all nodes nd such that nd is an alternative equal node of some node in S_i constructed from some node in $\bigcup_{j=1}^{i-1} Comb_j(S)$.*

Then, $Comb(S) := \bigcup_{i=1}^{\infty} Comb_i(S)$ is called the set of combined nodes of S *and a node in $Comb_i(S)$ is called a* combined node of cardinality i in S.

Further, let node *be a node in* DYNAMICHS *and X be a minimal conflict set w.r.t. the current DPI. Then,*

- *$Comb_{\mathsf{node}}(S) := \{\mathsf{nd} \mid \mathsf{nd} \in Comb(S), \mathsf{nd} = \mathsf{node}\}$ is the set of combined equal nodes of nd of S and*

- *$Comb_{\mathsf{node}, X}(S) \subseteq Comb_{\mathsf{node}}(S)$ is the set of combined equal nodes of nd of S for which X is not a witness of redundancy.*

The following corollary summarizes some simple consequences of Definition 12.5.

Corollary 12.6. *Let S be a set of nodes in* DYNAMICHS *and let S_i be the set of nodes of cardinality i in S. Then:*

(1) $Comb_i(S) = \emptyset$ iff $S_i = \emptyset$.

(2) $Comb_i(S)$ includes only nodes of cardinality i.

(3) $Comb_{\mathsf{node}}(S) = \emptyset$ iff there is no node $\mathsf{nd} \in S$ such that $\mathsf{nd} = \mathsf{node}$.

(4) If $\mathsf{nd} \in Comb_i(S)$ and $\mathsf{nd} \notin S$, then

- *there is some $\mathsf{nd}' \in Comb_j(S)$ for some $j \in \{1, \ldots, i-1\}$ and*
- *some $\mathsf{nd}'' \in S_i$*

such that

- *nd' is an alternative subnode of nd'' and*
- *$\mathsf{nd} = \text{ADD}(\mathsf{nd}', \mathsf{nd}''[j+1..i])$ and*
- *$\mathsf{nd}.\mathsf{cs} = \text{ADD}(\mathsf{nd}'.\mathsf{cs}, \mathsf{nd}''.\mathsf{cs}[j+1..i])$.*

The example we give next illustrates Definition 12.5.

Example 12.6 Recall the nodes $\mathsf{nd}, \mathsf{nd}_1, \mathsf{nd}_2, \mathsf{node}_1$ and node_2 of Example 12.4 and let $\mathsf{nd}_3 := [1, 2, 6, 4]$ with $\mathsf{nd}_3.\mathsf{cs} := [\langle 1, 2, 3 \rangle, \langle 2, 6 \rangle, \langle 3, 6, 7 \rangle, \langle 4, 5 \rangle]$ and $S := \{\mathsf{nd}, \mathsf{nd}_1, \mathsf{nd}_2, \mathsf{nd}_3\}$. Then,

$$S_1 = \emptyset$$
$$S_2 = \{\mathsf{nd}_1\}$$
$$S_3 = \{\mathsf{nd}_2\}$$
$$S_4 = \{\mathsf{nd}, \mathsf{nd}_3\}$$
$$S_i = \emptyset \quad \forall i > 4$$
$$Comb_1(S) = \emptyset$$
$$Comb_2(S) = \{\mathsf{nd}_1\}$$
$$Comb_3(S) = \{\mathsf{nd}_2\}$$
$$Comb_4(S) = \{\mathsf{nd}, \mathsf{node}_1, \mathsf{node}_2, \mathsf{nd}_3, \mathsf{nd}_4\}$$
$$Comb_i(S) = \emptyset \quad \forall i > 4$$
$$Comb_{\mathsf{nd}}(S) = \{\mathsf{nd}, \mathsf{node}_1, \mathsf{node}_2\}$$
$$Comb_{\mathsf{node}_1}(S) = Comb_{\mathsf{node}_2}(S) = Comb_{\mathsf{nd}}(S)$$
$$Comb_{\mathsf{nd}_3}(S) = \{\mathsf{nd}_3, \mathsf{nd}_4\}$$
$$Comb_{\mathsf{nd}_4}(S) = Comb_{\mathsf{nd}_3}(S)$$

where

$$\mathsf{nd}_4 := [2, 1, 6, 4]$$
$$\mathsf{nd}_4.\mathsf{cs} := [\langle 1, 2, 3 \rangle, \langle 1, 4 \rangle, \langle 3, 6, 7 \rangle, \langle 4, 5 \rangle]$$

is the alternative equal node of nd_3 constructed from nd_1. $\square$

The PRUNEQDUP function is always called given the current list $\mathbf{Q}_{dup}$ which is anytime sorted in ascending order by node cardinality. This holds by lines 21, 121 and 124 which are the only places where nodes are added to $\mathbf{Q}_{dup}$ throughout DYNAMICHS and where nodes are inserted into $\mathbf{Q}_{dup}$ such that the order by node cardinality is preserved. Now, the next lemma substantiates that PRUNEQDUP, given some minimal conflict set X w.r.t. the current DPI, updates $\mathbf{Q}_{dup}$ in a way that all redundant nodes w.r.t. the witness X are deleted, each deleted node is replaced by one non-redundant combined replacement node w.r.t. the witness X if such a one is constructable (cf. Definition 12.5), and for each remaining node nd, i.e. nd is a non-deleted node or a combined replacement node of some deleted node, each superset of X in nd.cs is replaced by X.

This leads to a new list $\mathbf{Q}_{dup}$ returned by PRUNEQDUP which includes only non-redundant nodes w.r.t. the witness X. Furthermore, the new list $\mathbf{Q}_{dup}$ contains a node corresponding to each set (path) S for which there was a corresponding node in the old list $\mathbf{Q}_{dup}$ if there would be a non-redundant (w.r.t. X) node corresponding to S in a hitting set tree equal to the one produced by DYNAMICHS except that all duplicate nodes corresponding to equal sets (paths) would be regularly processed and expanded.

Lemma 12.8. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI and let the input parameters to the* PRUNEQDUP *function be:*

- *X is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$,*

- *Dup is a set of nodes sorted ascending by node cardinality.*

Then, PRUNEQDUP *returns Dup_{new} where Dup_{new} includes*

(1) all nodes in Dup for which X is not a witness of redundancy,

(2) at least one node in $Comb_{\mathsf{nd},X}(Dup)$ for each node $\mathsf{nd} \in Dup$ for which X is a witness of redundancy, if $Comb_{\mathsf{nd},X}(Dup) \neq \emptyset$ and

(3) only nodes nd such that there is no $r \in \{1, \ldots, |\mathsf{nd}|\}$ for which $\mathsf{nd.cs}[r] \supset X$.

Proof. The function PRUNEQDUP walks through all nodes ndi in the set Dup. If X is not a witness of redundancy of ndi, tested in lines 111 and 112 exactly as prescribed by Definition 12.4, then $k = 0$ must hold in line 116 by lines 109-115. Thus, line 124 is executed and ndi added to Dup_{new}. Since no nodes are removed from Dup_{new} throughout PRUNEQDUP, proposition (1) is valid.

Otherwise, i.e. if X is a witness of redundancy of ndi, then line 113 must have been executed at least once before line 116 is reached. This implies that $k > 0$ must hold in line 116. At this point, k stores the maximum position in (the list) ndi at which the redundancy criterion of lines 111 and 112 is satisfied. So, in line 117, nodes in Dup_{new} are tested successively until some $\mathsf{ndj} \in Dup_{new}$ meets $|\mathsf{ndj}| \geq k$ and $\mathsf{ndi}[1..|\mathsf{ndj}|] = \mathsf{ndj}$. This means that the subnode $\mathsf{ndi}[1..|\mathsf{ndj}|]$ of ndi can be replaced by ndj (and $\mathsf{ndi.cs}[1..|\mathsf{ndj}|]$ by ndj.cs) to yield an alternative equal node ndi_{new} of ndi (lines 119 and 120).

We still have to show that X cannot be a witness of redundancy of ndi_{new}. For this to hold it is sufficient that X is not a witness of redundancy of ndj by $|\mathsf{ndj}| \geq k$. So, we must verify that Dup_{new} can comprise only nodes of which X is not a witness of redundancy. We prove this by induction.

Since Dup_{new} is initialized to be the empty set when the function PRUNEQDUP starts executing, we just need to investigate which nodes are added to Dup_{new} within PRUNEQDUP. Addition of nodes to Dup_{new} happens at lines 121 and 124.

Base case: When line 121 executed for the first time during the execution of PRUNEQDUP, Dup_{new} can only comprise nodes which have been added to it in line 124. By the argumentation used to prove proposition (1) of this lemma, it holds that X is not a witness of redundancy of any node added to Dup_{new} in line 124. Thus, there cannot be a witness of redundancy of the very first node added to Dup_{new} in line 121.

Induction step: Let us assume that Dup_{new} comprises only nodes such that X is not a witness of redundancy of any of them. Further, suppose that ndi_{new} is added to Dup_{new} when line 121 is executed

for the k-th time where $k > 1$. Then, by the same line of argument as in the base case, we can conclude that X is not a witness of redundancy of ndi_{new}.

Each node ndi_{new} added to Dup_{new} in line 121 is an element of $Comb_{\mathsf{ndi},X}(Dup)$. Namely, ndj satisfies the criterion in line 118 and thus ndi_{new} is an element of $Comb_{\mathsf{ndi}}(Dup)$ by Definition 12.5. And, as shown before, X is not a witness of redundancy of ndi_{new}, wherefore $\mathsf{ndi}_{new} \in Comb_{\mathsf{ndi},X}(Dup)$ by the definition of $Comb_{\mathsf{ndi},X}(Dup)$ (Definition 12.5).

Thence, if $Comb_{\mathsf{ndi},X}(Dup) \neq \emptyset$, there must be at least one node nd added to Dup_{new} such that $\mathsf{nd} \in Comb_{\mathsf{ndi},X}(Dup)$ and X is not a witness of redundancy of nd. Consequently, proposition (2) holds.

Proposition (3): First, observe that each node in Dup is definitely processed as ndi by the for-loop in line 107 and the fact that there is no criterion that can cause a preliminary break of this for-loop. Each time the first part of the redundancy check (line 111) is successful for ndi, we know that some conflict set $\mathsf{ndi.cs}[m]$ is non-minimal w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. If the second part of the redundancy check (line 112) is negative, then $\mathsf{ndi}[m] \in X$, wherefore there is – at least so far – no evidence that ndi is redundant w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. In this case, ndi might later be inserted to Dup_{new} (in case X is not a witness of redundancy of ndi) and hence the set $\mathsf{ndi.cs}[m]$ is replaced by the minimal conflict set X w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ in line 115. If the second part of the redundancy check in line 112 is positive, then it is guaranteed that ndi is either combined-replaced or pruned. This holds due to lines 116-122 and since $k > 0$ must be true due to line 113. That a combined replacement node that might be found for some redundant ndi throughout lines 116-122 meets proposition (3) can be shown by induction in a very similar way as proposition (2) was shown. $\qquad\square$

The following corollary is a direct consequence of Lemma 12.8 and states that the updated list $\mathbf{Q}_{dup}$ (if interpreted as a set) is a subset of the set of combined nodes of the old list $\mathbf{Q}_{dup}$. In other words, no nodes corresponding to sets (paths) that are not represented by a node in the old list $\mathbf{Q}_{dup}$ can be introduced throughout PRUNEQDUP. The introduction of such nodes corresponding to "new" sets (paths) can only take place in line 21 where newly generated nodes are added to $\mathbf{Q}_{dup}$.

Corollary 12.7. *Given the same preconditions as in Lemma 12.8,* PRUNEQDUP *returns* Dup_{new} *where* $Dup_{new} \subseteq Comb(Dup)$.

The following result provides sufficient and necessary criteria for a node nd to be a combined node of $\mathbf{Q}_{dup}$. Roughly, these criteria involve the existence of a sequence of nodes $\mathsf{nd}_1, \ldots, \mathsf{nd}_k \in \mathbf{Q}_{dup}$ where each node in this sequence is a proper alternative subnode of the next node and nd is constructed from this sequence of nodes in that nd is an alternative equal node of nd_k constructed from nd'_{k-1}. nd'_{k-1} in turn is an alternative equal node of nd_{k-1} constructed from nd'_{k-2}, and so on. Finally, nd'_2 is an alternative equal node of nd_2 constructed from nd_1 and $\mathsf{nd}_1 \in \mathbf{Q}_{dup}$.

Lemma 12.9. *Let* nd *be a node in* DYNAMICHS. *Then,* $\mathsf{nd} \in Comb(\mathbf{Q}_{dup})$ *iff there are nodes* $\mathsf{nd}_1, \ldots,$ $\mathsf{nd}_k \in \mathbf{Q}_{dup}$ *for* $k \geq 1$ *such that*

(1) $|\mathsf{nd}_1| < \cdots < |\mathsf{nd}_k| = |\mathsf{nd}|$,

(2) it holds that

$$\mathsf{nd}[i_1] = \mathsf{nd}_1[i_1] \text{ for } i_1 \in \{1, \ldots, |\mathsf{nd}_1|\}$$
$$\mathsf{nd}[i_2] = \mathsf{nd}_2[i_2] \text{ for } i_2 \in \{|\mathsf{nd}_1| + 1, \ldots, |\mathsf{nd}_2|\}$$
$$\cdots$$
$$\mathsf{nd}[i_k] = \mathsf{nd}_k[i_k] \text{ for } i_k \in \{|\mathsf{nd}_{k-1}| + 1, \ldots, |\mathsf{nd}_k|\}$$

and

(3) nd_i *is an alternative subnode of* nd_{i+1} *for* $i \in \{1, \ldots, k-1\}$.

Proof. "$\Rightarrow$": Suppose $\mathsf{nd} \in Comb(\mathbf{Q}_{dup})$ and that $|\mathsf{nd}| = i$. Then, there are two cases, either $\mathsf{nd} \in \mathbf{Q}_{dup}$ or $\mathsf{nd} \notin \mathbf{Q}_{dup}$.

In the former case, we can define nd_1 as nd and the proposition of the lemma holds.

In the latter case, by proposition 2 of Corollary 12.6, Definition 12.5 and $|\mathsf{nd}| = i$, it holds that $\mathsf{nd} \in Comb_i(\mathbf{Q}_{dup})$. By proposition 4 of Corollary 12.6 and the fact that $\mathsf{nd} \in Comb_i(\mathbf{Q}_{dup})$, there is some $\mathsf{nd}' \in Comb_j(\mathbf{Q}_{dup})$ for some $j \in \{1, \dots, i-1\}$ and some $\mathsf{nd}'' \in \mathbf{Q}_{dup}$ with $|\mathsf{nd}''| = i$ such that $\mathsf{nd} = \text{ADD}(\mathsf{nd}', \mathsf{nd}''[j+1..i])$ and $\mathsf{nd.cs} = \text{ADD}(\mathsf{nd}'.\mathsf{cs}, \mathsf{nd}''.\mathsf{cs}[j+1..i])$. Moreover, proposition 4 of Corollary 12.6 states that nd' is an alternative subnode of nd''.

So, set nd_k to nd'' and nd_{k-1} to nd'. Then, we obtain that $|\mathsf{nd}_{k-1}| < |\mathsf{nd}_k|$ by $j < i$, that nd_{k-1} is an alternative subnode of nd_k and that $\mathsf{nd}[i_k] = \mathsf{nd}_k[i_k]$ for $i_k \in \{|\mathsf{nd}_{k-1}| + 1, \dots, |\mathsf{nd}_k|\}$ must be true. That is, propositions (1), (2) and (3) hold for nd_k and nd_{k-1}.

Now, again, there are two cases for nd_{k-1}, i.e. either $\mathsf{nd}_{k-1} \in \mathbf{Q}_{dup}$ or $\mathsf{nd}_{k-1} \notin \mathbf{Q}_{dup}$.

In the former case, we can define nd_1 as nd_{k-1} and the proposition of the lemma holds.

In the latter case, the same argumentation as for nd can be applied to show the existence of some nd_{k-2} that meets propositions (1), (2) and (3). Due to the fact that the cardinality of nd_{k-i-1} is strictly smaller than the cardinality of nd_{k-i} for all i and the fact that $Comb_1(\mathbf{Q}_{dup}) = \mathbf{Q}_{dup}$, the case $\mathsf{nd}_{k-m} \in \mathbf{Q}_{dup}$ must finally arise for some m.

"$\Leftarrow$": Suppose there are nodes $\mathsf{nd}_1, \dots, \mathsf{nd}_k \in \mathbf{Q}_{dup}$ such that propositions (1)-(3) are satisfied. Let $k = 1$. Then, by propositions (1) and (2) of this lemma, we have that nd is the same node as nd_1. Since $\mathsf{nd}_1 \in \mathbf{Q}_{dup}$ and by Definition 12.5, we have that $\mathsf{nd} \in Comb(\mathbf{Q}_{dup})$. So, the lemma holds for $k = 1$.

Now, assume that the lemma holds for $k = m$ for some natural number m. That is, assume that there is a node $\mathsf{nd} \in Comb(\mathbf{Q}_{dup})$ if there are nodes $\mathsf{nd}_1, \dots, \mathsf{nd}_m \in \mathbf{Q}_{dup}$ such that $|\mathsf{nd}_1| < \cdots < |\mathsf{nd}_m| = |\mathsf{nd}|$,

$$\mathsf{nd}[i_1] = \mathsf{nd}_1[i_1] \text{ for } i_1 \in \{1, \dots, |\mathsf{nd}_1|\}$$
$$\mathsf{nd}[i_2] = \mathsf{nd}_2[i_2] \text{ for } i_2 \in \{|\mathsf{nd}_1| + 1, \dots, |\mathsf{nd}_2|\}$$
$$\dots$$
$$\mathsf{nd}[i_m] = \mathsf{nd}_m[i_m] \text{ for } i_m \in \{|\mathsf{nd}_{m-1}| + 1, \dots, |\mathsf{nd}_m|\}$$

and nd_i is an alternative subnode of nd_{i+1} for $i \in \{1, \dots, m-1\}$.

Let now $k = m + 1$. That is, assume that there are nodes $\mathsf{nd}_1, \dots, \mathsf{nd}_{m+1} \in \mathbf{Q}_{dup}$ such that $|\mathsf{nd}_1| < \cdots < |\mathsf{nd}_{m+1}| = |\mathsf{nd}|$,

$$\mathsf{nd}'[i_1] = \mathsf{nd}_1[i_1] \text{ for } i_1 \in \{1, \dots, |\mathsf{nd}_1|\}$$
$$\mathsf{nd}'[i_2] = \mathsf{nd}_2[i_2] \text{ for } i_2 \in \{|\mathsf{nd}_1| + 1, \dots, |\mathsf{nd}_2|\}$$
$$\dots$$
$$\mathsf{nd}'[i_{m+1}] = \mathsf{nd}_{m+1}[i_{m+1}] \text{ for } i_{m+1} \in \{|\mathsf{nd}_m| + 1, \dots, |\mathsf{nd}_{m+1}|\}$$

and nd_i is an alternative subnode of nd_{i+1} for $i \in \{1, \dots, m\}$. What we need to show is that $\mathsf{nd}' \in Comb(\mathbf{Q}_{dup})$.

If $\mathsf{nd}' \in \mathbf{Q}_{dup}$, then, by Definition 12.5, the lemma is true. So suppose $\mathsf{nd}' \notin \mathbf{Q}_{dup}$.

By the definition of an alternative subnode (Definition 12.2), $\mathsf{node}_{sub} \subseteq \mathsf{node}$ in case node_{sub} is an alternative subnode of node. So, because nd_i is an alternative subnode of nd_{i+1} and $|\mathsf{nd}_i| < |\mathsf{nd}_{i+1}|$ for $i \in \{1, \dots, m\}$, we have that $\mathsf{nd}_i \subset \mathsf{nd}_{i+1}$ for $i \in \{1, \dots, m\}$. Consequently, $\mathsf{nd}_i \subseteq \mathsf{nd}'$ for $i \in \{1, \dots, m+1\}$ and $\mathsf{nd}_i \subseteq \mathsf{nd}$ for $i \in \{1, \dots, m\}$ must hold. Due to $|\mathsf{nd}_m| = |\mathsf{nd}|$ we obtain the set-equality between nd_m and nd. This result along with $\mathsf{nd}_m \subseteq \mathsf{nd}'$ and $|\mathsf{nd}_m| < |\mathsf{nd}_{m+1}| = |\mathsf{nd}'|$

implies that $\mathsf{nd} \subset \mathsf{nd}'$. However, since

$$\mathsf{ndx}[i_1] = \mathsf{nd}_1[i_1] \text{ for } i_1 \in \{1, \dots, |\mathsf{nd}_1|\}$$
$$\mathsf{ndx}[i_2] = \mathsf{nd}_2[i_2] \text{ for } i_2 \in \{|\mathsf{nd}_1| + 1, \dots, |\mathsf{nd}_2|\}$$
$$\dots$$
$$\mathsf{ndx}[i_m] = \mathsf{nd}_m[i_m] \text{ for } i_m \in \{|\mathsf{nd}_{m-1}| + 1, \dots, |\mathsf{nd}_m|\}$$

is met for ndx being the same node as nd as well as for ndx being the same node as nd', we can conclude that $\mathsf{nd}[i] = \mathsf{nd}'[i]$ for $i \in \{1, \dots, |\mathsf{nd}|\}$.

Moreover, we have that $\mathsf{nd}'[i_{m+1}] = \mathsf{nd}_{m+1}[i_{m+1}]$ for $i_{m+1} \in \{|\mathsf{nd}| + 1, \dots, |\mathsf{nd}'|\}$ since $|\mathsf{nd}| = |\mathsf{nd}_m|$ and $|\mathsf{nd}'| = |\mathsf{nd}_{m+1}|$.

Since $\mathsf{nd}' \notin \mathbf{Q}_{dup}$ and $\mathsf{nd}_{m+1} \in \mathbf{Q}_{dup}$ by assumption, we have that nd_{m+1} which is set-equal to nd' (as argued before) must be an alternative equal node of nd'. That is, there must be some j such that $\mathsf{nd}_{m+1}[j] \neq \mathsf{nd}'[j]$ or $\mathsf{nd}_{m+1}.\mathsf{cs}[j] \neq \mathsf{nd}'.\mathsf{cs}[j]$ for some $j \in \{1, \dots, |\mathsf{nd}|\}$. Hence, $\mathsf{nd}_{m+1}[j] \neq \mathsf{nd}[j]$ or $\mathsf{nd}_{m+1}.\mathsf{cs}[j] \neq \mathsf{nd}.\mathsf{cs}[j]$ wherefore nd must be an alternative subnode of nd_{m+1}. Because $\mathsf{nd} \in Comb(\mathbf{Q}_{dup})$ and $\mathsf{nd}_{m+1} \in \mathbf{Q}_{dup}$, we infer by Definition 12.5 that $\mathsf{nd}' \in Comb(\mathbf{Q}_{dup})$. $\square$

The PRUNE function (lines 63-65) is called given a collection $S \in \{\mathbf{Q}, \mathbf{D}_\times, \mathbf{D}_\supset\}$, a minimal conflict set X w.r.t. the current DPI and $\mathbf{Q}_{dup}$ which has already been updated and cleaned from redundant nodes (w.r.t. the witness X) by the PRUNEQDUP function. So, let $\mathsf{nd}_{dup} \in \mathbf{Q}_{dup}$ be a (not necessarily proper) alternative subnode of some node node that is stored in S. Assume X is a witness of redundancy of node. By Lemma 12.8 and since $\mathsf{nd}_{dup} \in \mathbf{Q}_{dup}$, X cannot be a witness of redundancy of nd_{dup}. Further, let $r \in \{1, \dots, |\mathsf{node}|\}$ be the highest number such that $X \subset \mathsf{node}.\mathsf{cs}[r]$ and $\mathsf{node}[r] \in \mathsf{node}.\mathsf{cs}[r] \setminus X$. Now, in case $r \leq |\mathsf{nd}_{dup}|$ holds, nd_{dup} (and $\mathsf{nd}_{dup}.\mathsf{cs}$) can be used to replace the first $|\mathsf{nd}_{dup}|$ elements of node (and $\mathsf{node}.\mathsf{cs}$). The result is an alternative equal node of node which is non-redundant w.r.t. the current DPI and which can be added to S after deletion of node as a representative of the set (path) node has represented.

Now, the next lemma substantiates that PRUNE updates S in a way that all redundant nodes w.r.t. the witness X are deleted, each deleted node is replaced by one non-redundant replacement node w.r.t. the witness X if such a one is constructable from $\mathbf{Q}_{dup}$ and for each remaining node nd, i.e. nd is a non-deleted node or a replacement node of some deleted node, each superset of X in $\mathsf{nd}.\mathsf{cs}$ is replaced by X.

This leads to a new set S returned by PRUNE which includes only non-redundant nodes w.r.t. the witness X. Furthermore, the new set S contains a node corresponding to each set (path) Y for which there was a corresponding node in the old set S if there would be a non-redundant (w.r.t. X) node corresponding to Y in a hitting set tree equal to the one produced by DYNAMICHS except that all duplicate nodes corresponding to equal sets (paths) would be regularly processed and expanded.

Lemma 12.10. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be a DPI and let the following be the input parameters to the* PRUNE *function:*

- *X is a minimal conflict set w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$,*

- *S is a set of nodes in* DYNAMICHS,

- *Dup is a set of nodes where*

   - *X is not a witness of redundancy of any node in Dup and*

   - *for each $\mathsf{nd} \in S$ there might be some $\mathsf{nd}' \in Dup$ such that nd' is an alternative subnode of nd and*

– *for each node* nd $\in Dup$ *there is no* $r \in \{1, \ldots, |\text{nd}|\}$ *for which* nd.cs$[r] \supset X$.

• p_{nodes} *is as defined by Definition 4.9.*

Then, PRUNE *returns* S' *where the following holds:*

(1) S' *is a set such that* $S \setminus S'$ *includes exactly these nodes in* S *for which* X *is a witness of redundancy and* $S \cap S'$ *includes exactly these nodes in* S *for which* X *is not a witness of redundancy.*

(2) *Each element* nd $\in S' \setminus S$ *is an alternative equal node of some node in* $S \setminus S'$ *constructed from some node in* Dup *such that* X *is not a witness of redundancy of* nd.

(3) *Let* nd $\in S \setminus S'$ *and* Alt_{nd} *denote the set of all alternative equal nodes of* nd*, each of which can be constructed from some node in* Dup *and for each of which* X *is not a witness of redundancy. Then there is some* nd$' \in Alt_{\text{nd}}$ *such that* nd$' \in S' \setminus S$.

(4) S' *includes only nodes* nd *such that there is no* $r \in \{1, \ldots, |\text{nd}|\}$ *for which* nd.cs$[r] \supset X$.

Proof. The PRUNE procedure runs through all nodes nd $\in S$ and for each nd runs through all sets in nd.cs (lines 87 and 89). Lines 90 and 91 perform a check whether X is a witness of redundancy of nd, implementing exactly the criteria given by Definition 12.4. If the check is not successful for any $i \in \{1, \ldots, |\text{nd}|\}$, i.e. X is not a witness of redundancy of nd, then $k = 0$ must hold when line 95 is reached. Hence, nd is added to S' in line 103 in this case. As only nodes different from nd can be added to S' in line 100 and as there are no other ways nodes might be added to S', we have that $S \setminus S'$ includes exactly these nodes in S for which X is a witness of redundancy and $S \cap S'$ includes exactly these nodes in S for which X is not a witness of redundancy. So, proposition (1) is true.

The truth of proposition (2) can be derived as follows: By the proof of proposition (1), line 100 is the only place where nodes that are not elements of S are added to S'. Hence, each node in $S' \setminus S$ must be added to S' in line 100. Thus, only nodes node$_{new}$:= ADD(node, nd$[|\text{node}| + 1..|\text{nd}|]$) with node$_{new}$.cs := ADD(node.cs, nd.cs$[|\text{node}| + 1..|\text{nd}|]$) constructed exactly as per Definition 12.2 in lines 98 and 99 where nd $\in S$ can be added to S'.

Now, we still have to show that node is an alternative subnode of nd. From the precondition that X is not a witness of redundancy of any node in Dup, X cannot be a witness of redundancy of node. Moreover, $|\text{node}| \geq k$ must hold as line 97 has been passed. So, we have that X must be a witness of redundancy for nd$[1..|\text{node}|]$ since $k > 0$ (line 95) and by the way k is constructed (lines 88-92). Hence, there must be some $j \in \{1, \ldots, |\text{node}|\}$ with the property that node$[j] \neq$ nd$[j]$ or node.cs$[j] \neq$ nd.cs$[j]$ wherefore node is indeed an alternative subnode of nd. Thus, node$_{new}$ is an alternative equal node of nd by Definition 12.2.

That nd $\in S \setminus S'$ must be true can be explained as follows. By the argumentation to prove proposition (1) and (2) so far, we know that only nodes can be added to S' in line 100 and line 103 for which X is not a witness of redundancy. Moreover, we have shown that line 100 can only be reached for some node nd $\in S$ for which X is a witness of redundancy. Consequently, nd $\notin S'$ must hold.

That X is not a witness of redundancy of node$_{new}$ can be derived as follows: From the precondition that X is not a witness of redundancy of any node in Dup, X cannot be a witness of redundancy of node$_{new}[1..|\text{node}|]$ with node$_{new}$.cs$[1..|\text{node}|]$ since node$_{new}[j] =$ node$[j]$ and node$_{new}$.cs$[j] =$ node.cs$[j]$ for all $j \in \{1, \ldots, |\text{node}|\}$. k is the maximum index such that $X \subset$ nd.cs$[k]$ and nd$[k] \in$ nd.cs$[k] \setminus X$ by lines 88-92. Since $|\text{node}| \geq k$, X cannot be a witness of redundancy of node$_{new}[|\text{node}| + 1..|\text{nd}|]$ with node$_{new}$.cs$[|\text{node}| + 1..|\text{nd}|]$ either since node$_{new}[j] =$ nd$[j]$ and node$_{new}$.cs$[j] =$ nd.cs$[j]$ for all $j \in \{|\text{node}| + 1, \ldots, |\text{nd}|\}$. Therefore, X cannot be a witness of redundancy of node$_{new}$.

Proposition (3): As already argued, for each node nd $\in S \setminus S'$, line 96 must be reached. Then, in line 96, *all* nodes in Dup are investigated in order to find an alternative subnode of nd. So, if there is such a one, then it must be found.

Proposition (4): For a node nd that is added to S' in line 103, the for-loop in line 89 must have been executed. Since, as already shown, line 92 cannot be executed for a node that is added to S' in line 103, line 94 must have been executed for all $i \in \{1, \ldots, |\mathsf{nd}|\}$. Hence, proposition (4) holds for all nodes inserted into S' in line 103.

For nodes

$$\mathsf{node}_{new} := \text{ADD}(\mathsf{node}, \mathsf{nd}[|\mathsf{node}| + 1..|\mathsf{nd}|])$$
$$\mathsf{node}_{new}.cs := \text{ADD}(\mathsf{node}.cs, \mathsf{nd}.cs[|\mathsf{node}| + 1..|\mathsf{nd}|])$$

inserted into S' in line 100, proposition (4) follows from the precondition that Dup includes only nodes n such that there is no $r \in \{1, \ldots, |\mathsf{n}|\}$ for which $\mathsf{n}.cs[r] \supset X$, from the fact that $\mathsf{node} \in Dup$ and the fact that line 94 must have been executed for all indices $i > k$. □

12.4.7   De-Facto Non-Redundant Nodes in DYNAMICHS

The following definition introduces a notion that is of rather theoretical use for the proof of completeness of DYNAMICHS we will give later. The definition assumes a fixed DPI and characterizes as active sublabel of a particular conflict set $\mathsf{nd}.cs[r]$ in $\mathsf{nd}.cs$ the subset of $\mathsf{nd}.cs[r]$ that "survives" all the pruning steps, i.e. PRUNEQDUP and PRUNE calls, during all executions of DYNAMICHS up to the one with a current DPI DPI. Notice that the shape of the active sublabel can never be known in advance as we do not know which witnesses of redundancy might be found. This makes up the theoretical nature of this definition. However, we will be able to show that no active sublabel of a node can be the empty set under certain preconditions that are met for DYNAMICHS.

Definition 12.6. *Let*

- nd *be a node in* DYNAMICHS,

- $r \in \{1, \ldots, |\mathsf{nd}|\}$ *fixed,*

- $DPI_1, \ldots, DPI_n$ *be a sequence of DPIs where DPI_j includes a proper subset of the test cases DPI_{j+1} includes for $j \in \{1, \ldots, n-1\}$,*

- DPI_n *is equal to DPI or includes a proper subset of the test cases DPI includes,*

- $\mathcal{C}_1, \ldots, \mathcal{C}_n$ *be the chronological sequence of all sets X given as an argument to* PRUNE *and* PRUNEQDUP *during all executions of* DYNAMICHS *up to and including the one with current DPI DPI where*

  - *each $\mathcal{C}_i$ is a minimal conflict set w.r.t. DPI_i for $i \in \{1, \ldots, n\}$*
  - *$\mathcal{C}_k \supset \mathcal{C}_{k+1}$ for $k \in \{1, \ldots, n-1\}$,*
  - *$\mathsf{nd}.cs[r] \supset \mathcal{C}_1$.*

Then, we call $\mathcal{C}_n$ the active sublabel of $\mathsf{nd}.cs[r]$ w.r.t. DPI.

The next definition of a de-facto non-redundant node is based on Definition 12.6. A de-facto non-redundant node w.r.t. DPI includes at each position an element that hits the active sublabel w.r.t. DPI at this position. Again, this definition is of theoretical rather than practical use, but crucial for the proof of completeness of DYNAMICHS. In fact, we will be able to show that for each minimal diagnosis w.r.t. DPI there must be – anytime during any execution of DYNAMICHS with a current DPI including a subset of the test cases in DPI – a de-facto non-redundant node corresponding to a subset of this diagnosis. In further consequence, this will allow us to derive the algorithm's completeness concerning the detection of all minimal diagnoses w.r.t. DPI.

Definition 12.7. *We call a node* nd *in* DYNAMICHS *de-facto non-redundant w.r.t.* DPI *iff* nd$[r]$ *is an element of an active sublabel w.r.t.* DPI *for all* $r \in \{1, \ldots, |nd|\}$.

A de-facto non-redundant node w.r.t. a DPI DPI "survives" all pruning steps at least until the execution of DYNAMICHS with current DPI DPI:

Proposition 12.7. *Let* nd *be a node which is de-facto non-redundant w.r.t.* DPI. *Then,* nd *cannot be pruned or replaced during any execution of* DYNAMICHS *up to and including the one with current DPI* DPI.

Proof. By Definitions 12.6 and 12.7, PRUNE and PRUNEQDUP cannot be called given a witness of redundancy of nd during any execution of DYNAMICHS up to and including the one with current DPI DPI. By Lemmata 12.8 and 12.10, only nodes can be pruned or replaced for which the input set X given to PRUNE and PRUNEQDUP is a witness of redundancy. □

Example 12.7 Let $\mathcal{K} = \{1, \ldots, 10\}$ be the KB of the (admissible) input DPI DPI_0 to Algorithm 5 and let nd := $[1, 2, 3, 4]$ with nd.cs := $[\langle 1, 5, 7 \rangle, \langle 2, 4, 6 \rangle, \langle 3, 6, 7 \rangle, \langle 4, 5 \rangle]$ be a node stored by DYNAMICHS during the execution of some call to DYNAMICHS during Algorithm 5. Moreover, let DPI be a fixed DPI constructed during the execution Algorithm 5 that includes a (not necessarily proper) superset of the test cases in DPI_0. Assume that the chronological sequence of all inputs X to PRUNE and PRUNEQDUP throughout all executions of DYNAMICHS up to and including the one with current DPI DPI during Algorithm 5 and after nd has been generated is given by $\langle 1, 6 \rangle, \langle 3, 7 \rangle, \langle 1, 3, 8 \rangle, \langle 2 \rangle, \langle 4 \rangle, \langle 1, 5 \rangle$.

Then nd.cs undergoes the transition depicted by Table 12.1 induced by this sequence of X arguments to PRUNE/PRUNEQDUP. We can observe in Table 12.1 that each proper superset of some argument X of

X	nd.cs after PRUNE/PRUNEQDUP with argument X
$\langle 1, 6 \rangle$	$[\langle 1, 5, 7 \rangle, \langle 2, 4, 6 \rangle, \langle 3, 6, 7 \rangle, \langle 4, 5 \rangle]$
$\langle 3, 7 \rangle$	$[\langle 1, 5, 7 \rangle, \langle 2, 4, 6 \rangle, \langle 3, 7 \rangle, \langle 4, 5 \rangle]$
$\langle 1, 3, 8 \rangle$	$[\langle 1, 5, 7 \rangle, \langle 2, 4, 6 \rangle, \langle 3, 7 \rangle, \langle 4, 5 \rangle]$
$\langle 2 \rangle$	$[\langle 1, 5, 7 \rangle, \langle 2 \rangle, \langle 3, 7 \rangle, \langle 4, 5 \rangle]$
$\langle 4 \rangle$	$[\langle 1, 5, 7 \rangle, \langle 2 \rangle, \langle 3, 7 \rangle, \langle 4 \rangle]$
$\langle 1, 5 \rangle$	$[\langle 1, 5 \rangle, \langle 2 \rangle, \langle 3, 7 \rangle, \langle 4 \rangle]$

Table 12.1: Transition of nd.cs induced by multiple calls to PRUNE.

PRUNE/PRUNEQDUP that occurs in nd.cs is replaced by X (cf. Lemmata 12.8 and 12.10). This is the case, for instance, for $X = \langle 3, 7 \rangle$ in the second row of the table which replaces nd.cs$[3] = \langle 3, 6, 7 \rangle$. Similar situations can be found in rows 4-6. No changes to nd.cs are triggered for $X = \langle 1, 6 \rangle$ or $X = \langle 1, 3, 8 \rangle$ in rows 1 and 3, respectively, because at this stage nd.cs does not include any superset of X.

We learn from the last row of the table that nd is de-facto non-redundant w.r.t. DPI. This holds, first, since we considered the chronological sequence of *all* inputs X to PRUNE and PRUNEQDUP throughout *all* executions of DYNAMICHS up to and including the one with current DPI DPI during Algorithm 5. Second, we have that

$$
\begin{aligned}
\text{nd}[1] = 1 &\in \langle 1, 5 \rangle = \text{nd.cs}'[1] \\
\text{nd}[2] = 2 &\in \langle 2 \rangle = \text{nd.cs}'[2] \\
\text{nd}[3] = 3 &\in \langle 3, 7 \rangle = \text{nd.cs}'[3] \\
\text{nd}[4] = 4 &\in \langle 4 \rangle = \text{nd.cs}'[4]
\end{aligned}
$$

where nd.cs$'$ is the value of nd.cs given by the last row of the table which is the "current" value of nd.cs during the execution of DYNAMICHS with current DPI DPI. By Definition 12.6, nd.cs$'[i]$ is the active sublabel of nd.cs$[i]$ w.r.t. DPI for $i \in \{1, \ldots, 4\}$. That is, for example, $\langle 3, 7 \rangle$ is the active sublabel of nd.cs$[3]$. As we realized that each element of nd is an element of an active sublabel w.r.t. DPI, we obtain the de-facto non-redundancy of nd w.r.t. DPI as per Definition 12.7.

Notice that the sole definition of redundancy of a node w.r.t. DPI (Definition 12.4) does not perfectly serve our purposes as it does not take into account the order in which new conflict sets emerge and are used for pruning.

For instance, consider nd.cs$[2] = \langle 2, 4, 6 \rangle$ which includes 2 as well as 4. Both values $\langle 2 \rangle$ and $\langle 4 \rangle$ of X in rows 4 and 5 of Table 12.1 must be conflict sets w.r.t. DPI by Proposition 12.1, which says that conflict sets cannot grow after the addition of a test case to a DPI, and the fact that each X must be a minimal conflict set w.r.t. some DPI including a subset of the test cases in DPI. In fact, by Proposition 12.2 and the admissibility of DPI_0, $\langle 2 \rangle$ and $\langle 4 \rangle$ are even *minimal* conflict sets w.r.t. DPI. Thus, application of Definition 12.4 yields that nd is redundant w.r.t. DPI because $\langle 4 \rangle \subset \langle 2, 4, 6 \rangle$ and nd$[2] = 2 \in \langle 2, 4, 6 \rangle \setminus \langle 4 \rangle$ (cf. Definition 12.4). However, bearing in mind that $\langle 2 \rangle$ was known to the algorithm before $\langle 4 \rangle$, or, $\langle 2 \rangle$ was used for pruning before $\langle 4 \rangle$, we have that the set nd.cs$[2]$, after being modified by PRUNE or PRUNEQDUP, is not redundant w.r.t. DPI. This is true since the new set nd.cs$[2] = \langle 2 \rangle$ which is not a superset of $\langle 4 \rangle$.

So, to summarize, a node is (theoretically) redundant w.r.t. DPI as per Definition 12.4 iff there is a minimal conflict set w.r.t. DPI which is a witness of redundancy of this node. As however the example above has shown, whether a node is found to be redundant or not depends on the order of conflict sets used for pruning. This fact is also mentioned in [GSW89]. And, a (theoretically) redundant node w.r.t. DPI does not necessarily need to be discovered by DYNAMICHS and might be modified by PRUNE or PRUNEQDUP in a way that it becomes non-redundant w.r.t. DPI.

On the other hand, the definition of de-facto non-redundancy w.r.t. DPI (Definition 12.7) incorporates exactly these thoughts and declares only nodes as de-facto non-redundant w.r.t. DPI which are *actually* not found to be redundant w.r.t. DPI. □

The criteria for a node nd to be a combined node of $\mathbf{Q}_{dup}$ given by Lemma 12.9 will facilitate the proof of the next lemma. This lemma states that a combined node of $\mathbf{Q}_{dup}$ which is non-redundant w.r.t. some DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ cannot be pruned during DYNAMICHS given i.a. the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and sets of positively and negatively answered queries P'' and N'' as input where $P'' \subseteq P'$ and $N'' \subseteq N'$. This result will constitute an essential prerequisite for the proof of completeness of DYNAMICHS.

Lemma 12.11. *Let* nd $\in Comb(\mathbf{Q}_{dup})$ *be some node that is de-facto non-redundant w.r.t. the DPI DPI and let DPI' be some DPI that is either equal to DPI or includes only a subset of the test cases of DPI. Then, throughout any execution of* DYNAMICHS *using the current DPI DPI',* nd $\in Comb(\mathbf{Q}_{dup})$ *holds.*

Proof. First, we show that there cannot be a minimal conflict set $\mathcal{C}$ w.r.t. DPI' such that PRUNEQDUP is called with $X := \mathcal{C}$ and there is some $q \in \{1, \ldots, |$nd$|\}$ with the property that $\mathcal{C} \subset$ nd.cs$[q]$ and nd$[q] \in$ nd.cs$[q] \setminus \mathcal{C}$.

So, assume that PRUNEQDUP is called with $X := \mathcal{C}$ and there is some $\mathcal{C}$ w.r.t. DPI' such that there is some $q \in \{1, \ldots, |$nd$|\}$ with the property that $\mathcal{C} \subset$ nd.cs$[q]$ and nd$[q] \in$ nd.cs$[q] \setminus \mathcal{C}$. Let now $\mathcal{C}_1, \ldots, \mathcal{C}_n$ be the (arbitrary actual) chronological sequence of all sets X given as an argument to PRUNE and PRUNEQDUP during all executions of DYNAMICHS up to and including the one with current DPI DPI where

- nd.cs$[q] \supset \mathcal{C}_1$,

- each $\mathcal{C}_i$ is a minimal conflict set w.r.t. DPI_i for $i \in \{1, \ldots, n\}$

- $\mathcal{C}_k \supset \mathcal{C}_{k+1}$ for $k \in \{1, \ldots, n-1\}$,

- DPI_j includes a proper subset of the test cases DPI_{j+1} includes for $j \in \{1, \ldots, n-1\}$,

- DPI_n is equal to DPI or includes a proper subset of the test cases DPI includes.

Then, $\mathcal{C}_n$ is the active sublabel of nd.cs$[q]$ w.r.t. DPI. Since $\mathcal{C} \subset$ nd.cs$[q]$ and $X := \mathcal{C}$ is an argument of PRUNEQDUP during DPI', we have that $\mathcal{C}$ must be equal to some set $\mathcal{C}_j$ in the sequence $\mathcal{C}_1, \ldots, \mathcal{C}_n$. By Definition 12.7 and the de-facto non-redundancy of nd w.r.t. DPI, nd.cs$[q] \in \mathcal{C}_n$ must hold. By $\mathcal{C}_n \subseteq \mathcal{C}_j = \mathcal{C}$, we finally obtain nd.cs$[q] \in \mathcal{C}$, which is a contradiction to nd$[q] \in$ nd.cs$[q] \setminus \mathcal{C}$.

Lemma 12.9 and nd $\in Comb(\mathbf{Q}_{dup})$ guarantee the existence of nodes $\text{nd}_1, \ldots, \text{nd}_k \in \mathbf{Q}_{dup}$ for $k \geq 1$ such that

(1) $|\text{nd}_1| < \cdots < |\text{nd}_k| = |\text{nd}|$,

(2) it holds that

$$\text{nd}[i_1] = \text{nd}_1[i_1] \text{ for } i_1 \in \{1, \ldots, |\text{nd}_1|\}$$
$$\text{nd}[i_2] = \text{nd}_2[i_2] \text{ for } i_2 \in \{|\text{nd}_1| + 1, \ldots, |\text{nd}_2|\}$$
$$\cdots$$
$$\text{nd}[i_k] = \text{nd}_k[i_k] \text{ for } i_k \in \{|\text{nd}_{k-1}| + 1, \ldots, |\text{nd}_k|\}$$

and

(3) nd_i is an alternative subnode of nd_{i+1} for $i \in \{1, \ldots, k-1\}$.

So, let us assume that nd $\notin Comb(\mathbf{Q}_{dup})$ at some point in time during the execution of DYNAMICHS using the current DPI DPI'. That is, some node nd_j for some $j \in \{1, \ldots, k\}$ must have been deleted from $\mathbf{Q}_{dup}$. Nodes can only be deleted from $\mathbf{Q}_{dup}$ in the scope of the function PRUNEQDUP. By Lemma 12.8 and Corollary 12.2, only nodes for which X is a witness of redundancy can be deleted from $\mathbf{Q}_{dup}$ by the function PRUNEQDUP where X is the minimal conflict set given to PRUNEQDUP.

Thus, assume that nd_j for some $j \in \{1, \ldots, k\}$ is the first node among $\text{nd}_1, \ldots, \text{nd}_k \in \mathbf{Q}_{dup}$ deleted from $\mathbf{Q}_{dup}$ by PRUNEQDUP given the minimal conflict set X w.r.t. DPI' as an argument. Then, as X must be a witness of redundancy of nd_j, we have that there is some $m \in \{1, \ldots, |\text{nd}_j|\}$ such that $X \subset \text{nd}_j.\text{cs}[m]$ and $\text{nd}_j[m] \in \text{nd}_j.\text{cs}[m] \setminus X$.

Since Lemma 12.9 holds also for $j \leq k$ and nd_j is the first node among $\text{nd}_1, \ldots, \text{nd}_k \in \mathbf{Q}_{dup}$ deleted from $\mathbf{Q}_{dup}$, we deduce that there is some node node $\in Comb(\mathbf{Q}_{dup})$ such that $|\text{node}| = |\text{nd}_j|$ and $\text{node}[r] = \text{nd}[r]$ for $r \in \{1, \ldots, |\text{nd}_j|\}$ where $|\text{nd}_j| \leq |\text{nd}|$. As pointed out before, there cannot be any $q \in \{1, \ldots, |\text{nd}|\}$ such that $X \subset \text{nd}.\text{cs}[q]$ and $\text{nd}[q] \in \text{nd}.\text{cs}[q] \setminus X$. This, however, is a contradiction that there is some $m \in \{1, \ldots, |\text{nd}_j|\}$ such that $X \subset \text{nd}_j.\text{cs}[m]$ and $\text{nd}_j[m] \in \text{nd}_j.\text{cs}[m] \setminus X$.

Hence, none of the nodes $\text{nd}_1, \ldots, \text{nd}_k \in \mathbf{Q}_{dup}$ can be deleted throughout the execution of DYNAMICHS using the current DPI DPI'. Consequently, by Lemma 12.9, nd $\in Comb(\mathbf{Q}_{dup})$ must be preserved. $\square$

The finding of the next lemma is that a node nd in DYNAMICHS cannot be processed before all nodes that are set-equal to nd or proper subsets of nd have been generated.

Lemma 12.12. *Let $GenNodes$ be the set of all nodes generated throughout the execution of all calls to* DYNAMICHS *during the execution of Algorithm 5. Then, a node* nd *cannot be processed before each node* nd$' \in GenNodes$ *where* nd$' \subseteq$ nd *is generated.*

Proof. Let nd$' \in GenNodes$ such that nd$' \subseteq$ nd. Assume that nd is processed, but nd$'$ has not yet been generated. In order to be processed, nd must be an element of $\mathbf{Q}$. By the fact that nd$' \in GenNodes$, nd$'$ must be generated at some point in time. In order for nd$'$ to be generated, some node nd$''$ with nd$'' \subset$ nd$'$ must be an element of $\mathbf{Q}$. This follows from

- the fact that each generated node is a superset of some node in **Q** (cf. lines 6, 18 and 23 and Definition 12.3),

- the fact that **Q** can only be modified by (a) deleting from **Q** some node and adding a set of successor nodes of it to **Q** (lines 6, 7 and 23) or by (b) deleting from **Q** some node and possibly adding to **Q** a replacement node of it in the function PRUNE and

- the fact that for any replacement node nd_{rep} of nd it holds that $\mathsf{nd}_{rep} = \mathsf{nd}$.

By Lemma 4.14, each node which is a proper subset of another node has a higher probability as per $p_{nodes}()$. Since nd is processed before nd' is generated and nodes in **Q** are processed in descending order of $p_{nodes}()$ (lines 23 and 6), $p_{nodes}(\mathsf{nd}) > p_{nodes}(\mathsf{nd}'')$ where $\mathsf{nd}'' \subset \mathsf{nd}' \subseteq \mathsf{nd}$, contradiction. $\qquad\square$

The purpose of the following definition is to refer to a node that results from another node nd by several replacements conducted by PRUNE as a node in a transitive replaces-relation with nd. This will simplify the notation used in the following two lemmata.

Definition 12.8. *Let* $\mathsf{nd}_i \sim_{Rep} \mathsf{nd}_j$ *iff* nd_i *is a replacement node of* nd_j *computed so far by* PRUNE *at any time during the execution of any call to* DYNAMICHS *during the execution of Algorithm 5. Further, let the set* $Rep := \{ \langle \mathsf{nd}_i, \mathsf{nd}_j \rangle \mid \mathsf{nd}_i \sim_{Rep} \mathsf{nd}_j \}$. *Then we say that* nd_1 *is in a transitive replaces-relation with* nd_k *iff there is a sequence of nodes* $\mathsf{nd}_1, \mathsf{nd}_2, \ldots, \mathsf{nd}_{k-1}, \mathsf{nd}_k$ *such that* $\langle \mathsf{nd}_i, \mathsf{nd}_{i+1} \rangle \in Rep$ *for all* $i \in \{1, \ldots, k-1\}$.

12.4.8 Completeness of DYNAMICHS

Lemmata 12.13 and 12.14 constitute the key results towards proving the completeness of DYNAMICHS in terms of finding the complete set of minimal diagnoses w.r.t. any current DPI DPI in case the execution of DYNAMICHS with current DPI DPI terminates on account of $\mathbf{Q} = []$. In other words, if there are no more open nodes in the hitting set tree constructed by DYNAMICHS with current DPI DPI, all minimal diagnoses w.r.t. DPI have been labeled by $valid$ and are thus elements of the set $\mathbf{D}_{calc}$.

The completeness proof (Lemma 12.8) will be a proof by induction where Lemma 12.13 will serve to derive the base case of the induction, whereas Lemma 12.14 will be exploited to establish the induction step.

Lemma 12.13 assumes an arbitrary fixed "current" DPI DPI such that DYNAMICHS with this "current" DPI DPI returns due to $\mathbf{Q} = []$. Further on, it assumes an arbitrary minimal diagnosis $\mathcal{D}$ w.r.t. DPI and a de-facto non-redundant node nd w.r.t. DPI which is a proper subset of $\mathcal{D}$ generated anytime throughout all executions of DYNAMICHS during the execution of Algorithm 5 up to the one with the current DPI DPI.

Given these preconditions, the lemma establishes the existence of a node nd_{suc} that corresponds to a superset of nd and to a subset of $\mathcal{D}$, includes one element more than the set nd and is generated anytime throughout all executions of DYNAMICHS during the execution of Algorithm 5 up to the one with the current DPI DPI. Moreover, it states that the node nd'_{suc} set-equal to this generated node that is an element of $\mathbf{Q}$ cannot be pruned. However, it might be replaced. In case there is only one potential replacement node of nd'_{suc} constructable from (the combined nodes of) $\mathbf{Q}_{dup}$, this replacement node is de-facto non-redundant w.r.t. DPI. Any node $\mathsf{nd}'_{suc,rep}$ in a transitive replaces relation with nd'_{suc} cannot be pruned either. It might again be replaced. In case there is only one potential replacement node of $\mathsf{nd}'_{suc,rep}$ constructable from (the combined nodes of) $\mathbf{Q}_{dup}$, this replacement node is de-facto non-redundant w.r.t. DPI.

Figuratively, with respect to the hitting set tree constructed by DYNAMICHS, this lemma predicates the following: Let the hitting set tree produced by DYNAMICHS be completely constructed for an arbitrary DPI DPI. In case there is any tree branch whose edge labels correspond to a part of the minimal

diagnosis $\mathcal{D}$ w.r.t. DPI and which is known to be definitely not pruned during this tree construction, then this branch must be extended by one edge labeled by an element of $\mathcal{D}$ and this extended path is known to be definitely not pruned during this tree construction.

Notice that during this tree construction, in practice, we will generally never be able to say that a *concrete* branch corresponding to a partial minimal diagnosis will definitely not be pruned. For, this depends on the answers to queries submitted by the interacting user. Nevertheless, for the proof of completeness of DYNAMICHS, it suffices to just know that there is *any* such branch in the tree.

Lemma 12.13. *Assume the execution of* DYNAMICHS *with the current DPI DPI and assume that the execution stops due to* $\mathbf{Q} = []$. *Let*

- $GenNodes$ *be the set of all nodes generated throughout the execution of all calls to* DYNAMICHS *during the execution of Algorithm 5,*

- $\mathcal{D}$ *be some minimal diagnosis w.r.t.* DPI,

- nd $\in GenNodes$ *such that* nd *is de-facto non-redundant w.r.t.* DPI *and* nd $\subset \mathcal{D}$ *and*

Then there are nodes nd_{suc} *and* nd'_{suc} *such that the following holds:*

(1) nd $\subset \mathsf{nd}_{suc} \subseteq \mathcal{D}$.

(2) $|\mathsf{nd}_{suc}| = |\mathsf{nd}| + 1$.

(3) $\mathsf{nd}_{suc} \in GenNodes$.

(4) $\mathsf{nd}'_{suc} = \mathsf{nd}_{suc}$ *is an element of* $\mathbf{Q}$ *immediately after* nd_{suc} *has been generated.*

(5) If PRUNE *is called given a witness of redundancy of* nd'_{suc}, *then some replacement node of* nd'_{suc} *is found. If only one replacement node of* nd'_{suc} *is found, then this replacement node is de-facto non-redundant w.r.t.* DPI.

(6) Let $\mathsf{nd}'_{suc,rep}$ *be in a transitive replaces-relation with* nd'_{suc}. *If* PRUNE *is called given a witness of redundancy of* $\mathsf{nd}'_{suc,rep}$, *then some replacement node of* $\mathsf{nd}'_{suc,rep}$ *is found. If only one replacement node of* $\mathsf{nd}'_{suc,rep}$ *is found, then this replacement node is de-facto non-redundant w.r.t.* DPI.

Proof. Now, since nd $\in GenNodes$, we know that nd must be generated at some point in time during the execution of any call to DYNAMICHS during the execution of Algorithm 5. As the execution Ex_{curr} of the call to DYNAMICHS using DPI is assumed to terminate due to $\mathbf{Q} = []$ and no more nodes can be generated after $\mathbf{Q} = []$ (each generated node is constructed by extending a node in $\mathbf{Q}$), nd must be generated the latest during Ex_{curr}.

So, let us consider exactly the point in time when nd is generated. Since this point in time might not arise during the execution Ex_{curr} of DYNAMICHS, but during some execution Ex_{prev} taking place before Ex_{curr} which uses some "current" DPI which includes fewer test cases than the current DPI DPI of Ex_{curr}, we call the "current" DPI in Ex_{prev} in the following DPI_{prev}. That is, DPI_{prev} might be equal to DPI or comprise a subset of the test cases DPI includes.

First, we observe that *immediately* after nd has been generated, there is some node nd$' \in \mathbf{Q}$ such that nd$' = $ nd. If nd$'$ is not the same node as nd, then nd $\in \mathbf{Q}_{dup}$. This follows from lines 20-23.

Second, we have that nd$' \in \mathbf{Q}$ cannot be pruned before it is processed. In case nd$'$ is the same node as nd, this follows from Proposition 12.7 and the precondition that nd is de-facto non-redundant w.r.t. DPI. Notice that in this case nd $\in \mathbf{Q}$ cannot even be replaced (also by Proposition 12.7).

Otherwise, if nd$'$ is not the same node as nd, we argue as follows: Assume that nd$'$ is redundant w.r.t. DPI_{prev} and that the PRUNE function is called with arguments $\mathbf{Q}$, $\mathbf{Q}_{dup}$ and some minimal conflict set

X w.r.t. DPI_{prev} which is a witness of redundancy of nd'. Then, since nd is de-facto non-redundant w.r.t. DPI, since DPI_{prev} includes a subset of the test cases DPI comprises and by Proposition 12.7, nd cannot have been deleted from $\mathbf{Q}_{dup}$ during any pruning step. Thence, by Lemma 12.10, nd (or some other node set-equal to nd' for which X is not a witness of redundancy) must be constructed and added to $\mathbf{Q}$ in lines 96-101 during the execution of the PRUNE function.

That is, before any node set-equal to nd is processed, any number of calls to PRUNE with arguments $\mathbf{Q}$, $\mathbf{Q}_{dup}$ and some minimal conflict set X w.r.t. *any* DPI DPI_{prev} imply that $\mathbf{Q}$ includes some node that is set-equal to nd. Let us denote by node the node set-equal to nd that is finally processed.

There must be some execution of DYNAMICHS with some DPI (which might be equal to DPI or include a subset of the test cases in DPI) during which node is processed. This holds as the execution of DYNAMICHS with DPI is assumed to stop because of $\mathbf{Q} = [\,]$, since not all nodes set-equal to node can be pruned, as just argued before, and because the only alternative way, except for pruning, to achieve the deletion of a node from $\mathbf{Q}$ (line 7) is to process it. Let DPI_{prev} now be the "current" DPI of the execution of DYNAMICHS during which node is processed. Further, we denote the DPI considered by the immediate subsequent execution of DYNAMICHS by DPI_{prev+1}, and so on.

When node is processed, it is either

- (a) labeled by a set (DLABEL returns in line 40, 46 or 34) or

- (b) not labeled by a set (DLABEL returns in line 29 or 43).

Case (b): In this case, DLABEL returns either

- (i) *nonmin* or

- (ii) *valid*.

Case (i): By Lemma 12.1, node must be a non-minimal diagnosis w.r.t. DPI_{prev}. By line 15, node is then added to the set $\mathbf{D}_{\supset}$. $\mathbf{D}_{\supset}$ is never modified throughout Algorithm 5 and is given as an input argument to each subsequent call to DYNAMICHS by line 10 in Algorithm 5. During the execution of some subsequent call to DYNAMICHS using the DPI DPI_{prev+i} for $i \geq 1$, the set $\mathbf{D}_{\supset}$ might be modified by the UPDATETREE function (line 65 and lines 70-78) or in the DLABEL function (line 38) called for DPI_{prev+i}. Because $\mathsf{node} = \mathsf{nd}$ and nd is de-facto non-redundant w.r.t. DPI, we infer by the same argumentation as used above that $\mathsf{node} \in \mathbf{D}_{\supset}$ cannot be pruned, i.e. node considered as a set cannot be deleted from $\mathbf{D}_{\supset}$ in line 65 or line 38. The truth of this is supported by Corollary 12.1 and Lemmata 12.6 and 12.7 which say that PRUNE can only be called given some minimal conflict set X w.r.t. DPI_{prev+i}. So, after any number of calls to PRUNE, we have that either $\mathsf{node} \in \mathbf{D}_{\supset}$ or, otherwise, there is some node in $\mathbf{D}_{\supset}$ which is set-equal to node and which is in a transitive replaces-relation with node. We keep calling this (possibly replacement) node node in the following.

By Lemma 12.1, at the time node was processed, there must be some diagnosis $\mathcal{D}'$ w.r.t. DPI_{prev} such that $\mathcal{D}' \in \mathbf{D}_{calc}$ and $\mathsf{node} \supset \mathcal{D}'$. Additionally, by Lemma 12.1, the set $\mathbf{D}_{calc}$ computed during DYNAMICHS for some "current" DPI DPI_j comprises only diagnoses w.r.t. DPI_j. Now, we have $\mathsf{node} \subset \mathcal{D}$ since $\mathsf{nd} \subset \mathcal{D}$ and $\mathsf{node} = \mathsf{nd}$, and $\mathcal{D}' \subset \mathsf{node}$. That is, $\mathcal{D}' \subset \mathcal{D}$. By the precondition that $\mathcal{D}$ is a minimal diagnosis w.r.t. DPI, $\mathcal{D}'$ cannot be a diagnosis w.r.t. DPI. Thus, there cannot be any such $\mathcal{D}'$ in $\mathbf{D}_{calc}$ computed during DYNAMICHS for DPI.

All nodes in $\mathbf{D}_{calc}$ returned by some call to DYNAMICHS using DPI DPI_1 that are no diagnoses w.r.t. DPI_2, the extension of DPI_1 by a new query added as a positive or negative test case, are added to the set $\mathbf{D}_{\times}$ (and not to $\mathbf{D}_{\checkmark}$) in line 22 of Algorithm 5 and are thus no elements of the set $\mathbf{D}_{\checkmark}$ given as an argument to DYNAMICHS at the next call to DYNAMICHS. The elements of $\mathbf{D}_{\checkmark}$ given as an argument to

DYNAMICHS at the next call to DYNAMICHS using DPI_2 are definitely added to $\mathbf{Q}$ again in lines 79-80 as $\mathbf{D}_\checkmark$ is not modified elsewhere in DYNAMICHS before lines 79-80 are reached.

Therefore, we need to differentiate between two cases: Either

- (x1) $\mathcal{D}' \in \mathbf{D}_\times$ never holds for the input argument $\mathbf{D}_\times$ to any call to DYNAMICHS or

- (x2) $\mathcal{D}' \in \mathbf{D}_\times$ holds at least once for the input argument $\mathbf{D}_\times$ to some call to DYNAMICHS.

Case (x1): Since $\mathcal{D}' \in \mathbf{D}_{calc}$ holds after the execution of DYNAMICHS using DPI_{prev} stops, we have that $\mathcal{D}' \in \mathbf{D}_\checkmark$ must hold for the argument $\mathbf{D}_\checkmark$ given to DYNAMICHS using DPI_{prev+1}. After UPDATETREE returns during DYNAMICHS using DPI_{prev+1}, $\mathcal{D}' \in \mathbf{Q}$ holds as argued. Subsequently, $\mathcal{D}'$ might be added again to $\mathbf{D}_{calc}$ and then to $\mathbf{D}_\checkmark$ again in line 21 of Algorithm 5 and to $\mathbf{Q}$ again in line 80 during DYNAMICHS using DPI_{prev+2}, and so forth. But, when a test case is added to some DPI DPI_{prev+i} in Algorithm 5 that invalidates the diagnosis $\mathcal{D}'$ (yielding the DPI $DPI_{prev+i+1}$), $\mathcal{D}' \notin \mathbf{D}_{calc}$ is assumed to hold (otherwise it would be an element of $\mathbf{D}_\times$ against our assumption). Such a test case must be added sometime as argued above. By Proposition 12.3, $\mathcal{D}'$ cannot be a (minimal) diagnosis w.r.t. any DPI including a superset of the test cases in $DPI_{prev+i+1}$ either. Notice that the case $\mathcal{D}' \notin \mathbf{D}_{calc}$ can emerge in spite of the fact that $\mathcal{D}'$ is a minimal diagnosis w.r.t. DPI_{prev+i} because there may be minimal diagnoses w.r.t. DPI_{prev+i} that have a higher probability as per $p_{nodes}()$ than $\mathcal{D}'$. For $DPI_{prev+i+1}$ and all DPIs including more test cases than $DPI_{prev+i+1}$, $\mathcal{D}'$ cannot be added to $\mathbf{D}_{calc}$ anymore due to Lemma 12.1 since only *diagnoses* w.r.t. the currently used DPI can be added to $\mathbf{D}_{calc}$.

Case (x2): Here, $\mathcal{D}' \in \mathbf{D}_\times$ holds at least once for the input argument $\mathbf{D}_\times$ to some call to DYNAMICHS using the DPI DPI_{prev+i}. Then, DYNAMICHS using the DPI $DPI_{prev+i-1}$ must have returned a set $\mathbf{D}_{calc}$ including $\mathcal{D}'$ as otherwise $\mathcal{D}'$ cannot be added to $\mathbf{D}_\times$. Hence, $\mathcal{D}'$ must be a diagnosis w.r.t. $DPI_{prev+i-1}$ by Lemma 12.1. Since $\mathcal{D}'$ is added to $\mathbf{D}_\times$, it cannot be a diagnosis w.r.t. DPI_{prev+i}. This must hold

- by Remark 7.4,

- since the set added to $\mathbf{D}_\times$ in Algorithm 5 is exactly the set $\mathbf{D}_{out}$ returned by GETINVALIDDIAGS in line 19 of Algorithm 5 and

- $\mathbf{D}_{out} = \mathbf{D}^+(Q)$ in case the user answer $u(Q)$ to the query Q w.r.t. $\mathbf{D}_{calc}$ and $DPI_{prev+i-1}$ is *false* and $\mathbf{D}_{out} = \mathbf{D}^-(Q)$ otherwise (notice that $\mathbf{D}_{calc}$ is called $\mathbf{D}_\checkmark$ in Algorithm 5).

So, by Proposition 12.3, $\mathcal{D}'$ cannot be a (minimal) diagnosis w.r.t. any DPI including more test cases than DPI_{prev+i} either.

Each element in $\mathbf{D}_\times$ is processed by the UPDATETREE function (lines 48-69) called for the DPI DPI_{prev+i}. In lines 48-69, each node ndx in $\mathbf{D}_\times$ can only be pruned or either ndx or a node in a transitive replaces-relation with ndx is added to $\mathbf{Q}$ in line 68. $\mathbf{D}_{calc}$ is not modified by UPDATETREE and $\mathbf{D}_{calc} = \emptyset$ holds at the beginning of the execution of each call to DYNAMICHS. (A node set-equal to) $\mathcal{D}'$ cannot ever be readded to $\mathbf{D}_{calc}$ by Lemma 12.1 and since $\mathcal{D}'$ is not a diagnosis w.r.t any DPI including more test cases than DPI_{prev+i}. Hence, $\mathcal{D}' \in \mathbf{D}_{calc}$ can never hold for any DPI including more test cases than DPI_{prev+i}.

Hence, there must be some DPI DPI_{prev+k} such that $\mathbf{D}_\checkmark$ given as input to the DYNAMICHS-call for DPI_{prev+k} does not include *any* diagnosis $\mathcal{D}' \subset$ node. So, during the execution of the call to DYNAMICHS using DPI DPI_{prev+k}, node must be deleted from $\mathbf{D}_\supset$ and be reinserted into $\mathbf{Q}$ by lines 70-78 in UPDATETREE which is called at the beginning of the execution of DYNAMICHS at any call to DYNAMICHS. This must hold since all nodes ndx in $\mathbf{D}_\supset$ that have not yet been pruned and for which there is no diagnosis in $\mathbf{D}_\checkmark$ which is a proper subset of ndx, are added to $\mathbf{Q}$ throughout lines 70-78.

As shown, both criteria are met for node during the execution of the call to DYNAMICHS using DPI DPI_{prev+k}.

Case (ii): By Lemma 12.1, we know that node is a diagnosis w.r.t. DPI_{prev} and that node is added to $\mathbf{D}_{calc}$. Since node $\subset \mathcal{D}$ and $\mathcal{D}$ is a minimal diagnosis w.r.t. DPI, we obtain, by the same argumentation as in (i), that there must be some DPI DPI_{prev+k} such that $\mathbf{D}_{\checkmark}$ given as input to the DYNAMICHS-call for DPI_{prev+k} does not include node.

If node $\notin \mathbf{D}_{\times}$, then it cannot ever be added to $\mathbf{D}_{calc}$ again, as argued in case (i). Otherwise, during the execution of UPDATETREE which is called at the beginning of the execution of each call to DYNAMICHS, $\mathbf{D}_{\times}$ is modified in lines 48-69.

Now, we differentiate between two cases, namely node is either

- ($\neg$r) non-redundant w.r.t. DPI or

- (r) redundant w.r.t. DPI.

Case ($\neg$r): Due to the non-redundancy of node w.r.t. DPI, Lemma 12.4, Lemma 12.10 and Corollary 12.1, node cannot be replaced or pruned throughout lines 48-66. Thus, node is reinserted into $\mathbf{Q}$ in line 68.

Case (r): Since node is redundant w.r.t. DPI, it may or may not be redundant w.r.t. $DPI_{prev+k+1}$. So, during the UPDATETREE function called in DYNAMICHS for $DPI_{prev+k+1}$, there may or may not be some call to PRUNE given some X as argument which is a witness of redundancy of node. In the latter case, node will not be replaced or pruned during any PRUNE execution and will be reinserted into $\mathbf{Q}$ in line 68. In the former case, node might be replaced, but it cannot be pruned due to the same reasoning as given above in case (i). So, either node or some node in a transitive replaces-relation with node must be in $\mathbf{D}_{\times}$ at the time line 67 is reached. This node is then added to $\mathbf{Q}$ in line 68.

Now, both cases (i) and (ii) identified for case (b) lead to the reinsertion of node or some node set-equal to node into $\mathbf{Q}$. Notice that this node has the same properties as node before one of the cases (i) or (ii) emerged. That is, if PRUNE is called given a witness of redundancy of node, then a replacement node of node is found. And, if only one replacement node of node is found, this replacement node is de-facto non-redundant w.r.t. DPI.

If node is the same node as nd, this holds since there cannot be a witness of redundancy of nd due to the de-facto non-redundancy of nd w.r.t. DPI and Proposition 12.7. Otherwise, this holds by Lemma 12.10 and since node = nd and nd $\in \mathbf{Q}_{dup}$ must hold due the de-facto non-redundancy of nd w.r.t. DPI and Proposition 12.7. So, we call this reinserted node again node.

Furthermore, node can be neither labeled by *valid* nor by *nonmin* during the execution of DYNAMICHS for DPI. This holds by Lemma 12.1 and since node can be neither a diagnosis nor a non-minimal diagnosis w.r.t. DPI due to node $\subset \mathcal{D}$ and the fact that $\mathcal{D}$ is a minimal diagnosis w.r.t. DPI. As a consequence of this and the assumption that the DYNAMICHS-call for DPI terminates due to $\mathbf{Q} = []$, case (a) must arise at some point in time for node during some execution of DYNAMICHS for some (previous) DPI not-necessarily equal to DPI.

Case (a): In this case, by Lemma 12.2, DLABEL returns a minimal conflict set L w.r.t. DPI_{prev} as a label for node where L has the property that $L \cap$ node $= \emptyset$.

It must hold that $L \neq \emptyset$. Otherwise, by Proposition 4.2, either

- (v1) $\mathcal{K}$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P_{prev}, N_{prev} \rangle_R$ where $DPI_{prev} = \langle \mathcal{K}, \mathcal{B}, P_{prev}, N_{prev} \rangle_R$ or

- (v2) DPI_{prev} is non-admissible.

In the former **case (v1)**, we know by Corollary 3.3 that the only (minimal) diagnosis w.r.t. DPI_{prev} is $\emptyset$. If DPI_{prev} is equal to DPI, this is a contradiction to the existence of some minimal diagnosis w.r.t. DPI, namely $\mathcal{D}$, which is not the empty set. $\mathcal{D} \supset \emptyset$ must hold since, by precondition, there is a node nd such that nd $\subset \mathcal{D}$ and since $\emptyset \subseteq$ nd.

Otherwise, if DPI_{prev} includes a proper subset of the test cases DPI includes, DPI can never be a current DPI during any execution of DYNAMICHS during the same execution of Algorithm 5 during which there is an execution of DYNAMICHS using DPI_{prev} as a current DPI. This holds as there must be at least two diagnoses in $\mathbf{D}_{\checkmark}$ (which is the set $\mathbf{D}_{calc}$ returned by DYNAMICHS for DPI_{prev}) in line 13 of Algorithm 5 in order for DYNAMICHS to be called again with an extended DPI. For, in case there is only one diagnosis, i.e. $\emptyset$, then the probability of this diagnosis is 1 which is greater or equal $1 - \sigma$ for any choice of σ due to $\sigma \geq 0$. Consequently, Algorithm 5 would return in line 14. This is a contradiction to the assumption that there is an execution of DYNAMICHS using DPI as a current DPI.

In the latter **case (v2)**, we can infer by Corollary 7.3, which states that adding queries as test cases to an admissible DPI can never yield a non-admissible DPI, that the DPI given as an input to Algorithm 5 must be non-admissible, contradiction.

Thence, $L \neq \emptyset$ and DYNAMICHS will execute lines 17-23 and generate one node $\text{node}_e := \text{ADD}(\text{node}, e)$ with $\text{node}_e.\text{cs} := \text{ADD}(\text{node.cs}, L)$ for each $e \in L$ (cf. Definition 12.2 for an explanation of the function ADD).

Now, we have that there must be some non-empty active sublabel of $L = \text{node}_e.\text{cs}[r]$ w.r.t. DPI where $r := |\text{node}_e|$ by Definition 12.6. This holds by the following argumentation:

The first observation is that $\text{node}_e.\text{cs}[r]$ cannot be reduced twice during one and the same execution of DYNAMICHS using one and the same DPI DPI_{prev+j} which results from DPI_{prev} by addition of test cases. For, by Corollaries 12.1 and 12.2 and Lemmata 12.6 and 12.7, PRUNE as well as PRUNEQDUP can only be called given some minimal conflict set X w.r.t. DPI_{prev+j}. By Lemmata 12.10 and 12.8, all nodes ndx that are in the set returned by PRUNE and PRUNEQDUP, respectively, have the property that there are no proper supersets of X in ndx.cs. Moreover, there are no proper subsets of X in ndx.cs. Because each ndx.cs[m] for $m \in \{1, \ldots, |\text{ndx.cs}|\}$ must be a minimal conflict set w.r.t. some DPI equal to DPI_{prev+j} or including a subset of the test cases in DPI_{prev+j}. Otherwise, ndx could not be a node during the execution of DYNAMICHS where DPI_{prev+j} is the current DPI. By Proposition 12.1, there cannot be any $m \in \{1, \ldots, |\text{ndx.cs}|\}$ such that ndx.cs[m] $\subset X$ as X is a minimal conflict set w.r.t. DPI_{prev+j}. As two minimal conflict sets w.r.t. DPI_{prev+j} can never be in a proper subset-relationship with one another, $L = \text{node}_e.\text{cs}[r]$ can be modified at most once by PRUNE or PRUNEQDUP for the DPI DPI_{prev+j}.

Second, by Proposition 12.1, each minimal conflict set w.r.t. DPI_{prev} is a conflict set w.r.t. any DPI DPI_{prev+j} that results from DPI_{prev} by addition of test cases, that is, in particular, w.r.t. DPI. So, there must be some minimal conflict set $\mathcal{C}_j$ w.r.t. each DPI_{prev+j} such that $\mathcal{C}_j \subseteq L$ and there cannot be any minimal conflict set w.r.t. DPI_{prev+j} that is a proper superset of L.

Third, we have that $L \neq \emptyset$, L is a minimal conflict set w.r.t. DPI_{prev}, and DPI_{prev+j} includes a superset of the test cases in DPI_{prev}. Thus, by Proposition 12.2, each minimal conflict set w.r.t. DPI_{prev+j} must be non-empty. In particular, this implies that all minimal conflict sets w.r.t. DPI that are subsets of L must be non-empty.

By these three observations, the criteria of Definition 12.6 can be applied to analyze the active subnode of $\text{node}_e.\text{cs}[r]$ w.r.t. DPI. That is, if $\mathcal{C}_1, \ldots, \mathcal{C}_n$ is the (arbitrary actual) chronological sequence of all sets X given as an argument to PRUNE and PRUNEQDUP during all executions of DYNAMICHS from the one with current DPI DPI_{prev} up to and including the one with current DPI DPI where

- $\text{node}_e.\text{cs}[r] \supset \mathcal{C}_1$,

- each $\mathcal{C}_i$ is a minimal conflict set w.r.t. DPI_i for $i \in \{1, \ldots, n\}$

- $\mathcal{C}_k \supset \mathcal{C}_{k+1}$ for $k \in \{1, \ldots, n-1\}$,

- DPI_j includes a proper subset of the test cases DPI_{j+1} includes for $j \in \{1, \ldots, n-1\}$,

- DPI_n is equal to DPI or includes a proper subset of the test cases DPI includes and

- DPI_{prev} includes a proper subset of the test cases DPI_1 includes,

then $\mathcal{C}_n$ is the active sublabel of $\mathsf{node}_e.\mathsf{cs}[r]$ w.r.t. DPI. However, as argued before, the minimal conflict set $\mathcal{C}_n$ w.r.t. DPI_n cannot be the empty set. As a consequence, we obtain that there must be a non-empty active sublabel of $\mathsf{node}_e.\mathsf{cs}[r]$ w.r.t. DPI.

By Propositions 12.1 and 12.2, there is a non-empty minimal conflict set $\mathcal{C}'$ w.r.t. DPI such that $\mathcal{C}' \subseteq \mathcal{C}_n$. Due to $\mathcal{C}_n \subset \cdots \subset \mathcal{C}_1 \subset \mathsf{node}_e.\mathsf{cs}[r] = L$ we conclude that $\mathcal{C}_n \subset L$. Therefore, $\emptyset \subset \mathcal{C}' \subset L$ holds.

By Proposition 4.6, each minimal diagnosis w.r.t. DPI is a minimal hitting set of all minimal conflict sets w.r.t. DPI. Thence, we have that $\mathcal{C}' \cap \mathcal{D} \neq \emptyset$. So, by $\mathcal{C}' \subset L$, we have that $\emptyset \subset \mathcal{C}' \cap \mathcal{D} \subseteq L \cap \mathcal{D} \subseteq L$. Consequently, we define $\mathsf{nd}_{suc} := \mathsf{node}_x = \textsc{add}(\mathsf{node}, x)$ with $\mathsf{nd}_{suc}.\mathsf{cs} := \mathsf{node}_x.\mathsf{cs} = \textsc{add}(\mathsf{node.cs}, L)$ for some $x \in \mathcal{C}' \cap \mathcal{D} \subseteq L$. Then, $\mathsf{nd}_{suc} \subseteq \mathcal{D}$ because $\mathsf{node} \subset \mathcal{D}$ and $x \in \mathcal{D}$. It is clear from the inference so far that $\mathsf{nd} \subset \mathsf{nd}_{suc}$, $|\mathsf{nd}_{suc}| = |\mathsf{nd}| + 1$ and $\mathsf{nd}_{suc} \in GenNodes$. This shows the truth of propositions (1)-(3).

Proposition (4) must hold by lines 20-23.

Now we argue why propositions (5) and (6) must hold. Assume that $\mathsf{nd}'_{suc} \in \mathbf{Q}$ is redundant w.r.t. some DPI DPI''_{prev} which is equal to DPI or includes a subset of the test cases in DPI. Then, there must be some minimal conflict set $\mathcal{C}''$ w.r.t. DPI''_{prev} which is a witness of redundancy of nd'_{suc}. Suppose that $\textsc{prune}$ is called given $X := \mathcal{C}''$ as an argument.

Now, we have to distinguish two cases: Either

- (q1) nd_{suc} was added to $\mathbf{Q}$ after it was generated or

- (q2) nd_{suc} was added to $\mathbf{Q}_{dup}$ after it was generated

(there are no other possibilities, see lines 17-23).

For each of these two cases, there are two more cases to discriminate between:

- (c1) $\mathcal{C}'' \subset \mathsf{nd}'_{suc}.\mathsf{cs}[|\mathsf{nd}'_{suc}|]$ and $\mathsf{nd}'_{suc}[|\mathsf{nd}'_{suc}|] \in \mathsf{nd}'_{suc}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] \setminus \mathcal{C}''$ or

- (c2) $\mathcal{C}'' \subset \mathsf{nd}'_{suc}.\mathsf{cs}[j]$ and $\mathsf{nd}'_{suc}[j] \in \mathsf{nd}'_{suc}.\mathsf{cs}[j] \setminus \mathcal{C}''$ for some $j \in \{1, \ldots, |\mathsf{nd}'_{suc}| - 1\}$.

Case (q1):   Here, we have that nd'_{suc} is the same node as nd_{suc} since nd_{suc} was added to $\mathbf{Q}$ after generation and no node replacement can have taken place because nd'_{suc} is defined as the node set-equal to nd_{suc} that is an element of $\mathbf{Q}$ *immediately* after nd_{suc} has been generated. And, only one node corresponding to one and the same set can be in $\mathbf{Q}$ at the same time.

Case (c1):   We have that $\mathcal{C}''$ must be equal to some minimal conflict set $\mathcal{C}_j$ in the sequence $\mathcal{C}_1, \ldots, \mathcal{C}_n$. This must be *true* since, first, DPI''_{prev} is equal to DPI or includes a subset of the test cases in DPI and DPI_{prev} includes a proper subset of the test cases in DPI''_{prev}.

To understand why the latter must hold, recall that DPI_{prev} is the DPI of the call to DYNAMICHS where nd_{suc} was generated and the minimal conflict set L was computed. By assumption, however, there is some minimal conflict set w.r.t. DPI''_{prev}, namely $\mathcal{C}''$, such that $\mathcal{C}'' \subset \mathsf{nd}'_{suc}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] = L$. Hence, it cannot be *true* that both L and $\mathcal{C}''$ are minimal conflict sets w.r.t. the same DPI. Otherwise, we would have a contradiction to the minimality of L. By Proposition 12.1, which states that minimal conflict sets cannot grow by the addition of new test cases to the DPI, we obtain the claimed fact that DPI_{prev} includes a proper subset of the test cases in DPI''_{prev}.

Second, the sequence $\mathcal{C}_1, \ldots, \mathcal{C}_n$ comprises *all* sets X given as an argument to PRUNE and PRUNEQDUP during *all* executions of DYNAMICHS from the one with current DPI DPI_{prev} up to and including the one with current DPI DPI where $\mathsf{nd}'_{suc}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] \supset \mathcal{C}_1 \supset \cdots \supset \mathcal{C}_n$ holds. Reason for this to be valid is the fact that nd'_{suc} is the same node as nd_{suc} in the currently considered case (q1).

Now, recall that $\mathcal{C}'$ is a minimal conflict set w.r.t. DPI such that $x \in \mathcal{C}' \cap \mathcal{D} \subset L$. Further, by $\mathsf{nd}'_{suc} = \mathsf{node}_x$, we have that $\mathsf{nd}'_{suc}[|\mathsf{nd}'_{suc}|] = x$. Due to $\mathcal{C}' \subseteq \mathcal{C}_n$ and $\mathcal{C}_n \subseteq \mathcal{C}_j$, we have that $\mathcal{C}' \subseteq \mathcal{C}_j$. Therefore, we can infer by $\mathcal{C}'' = \mathcal{C}_j$ that $\mathcal{C}' \subseteq \mathcal{C}''$ is *true*. Now, $x \in \mathcal{C}'$ implies that $x \in \mathcal{C}''$ wherefore $x \notin \mathsf{nd}'_{suc}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] \setminus \mathcal{C}''$. By $x = \mathsf{nd}'_{suc}[|\mathsf{nd}'_{suc}|]$, this is a contradiction to the assumption of case (c1). Hence, case (c2) must arise.

Case (c2): We have that $\mathsf{nd}'_{suc}[1..|\mathsf{nd}'_{suc}| - 1]$ is the same node as node since $\mathsf{nd}'_{suc} = \mathsf{node}_x$. Then, there are two cases: Either

- (s1) node is the same node as nd or

- (s2) node is not the same node as nd.

Case (s1): If node is the same node as nd, then node is de-facto non-redundant w.r.t. DPI since nd is de-facto non-redundant w.r.t. DPI by precondition. Moreover, x is an element of the active sublabel of $\mathsf{nd}'_{suc}[|\mathsf{nd}'_{suc}|]$ w.r.t. DPI, as specified before. Thus, by Definition 12.7, nd'_{suc} is de-facto non-redundant w.r.t. DPI. Hence, PRUNE cannot be given an argument $\mathcal{C}''$ which is a witness of redundancy of nd'_{suc} where $\mathcal{C}''$ is a minimal conflict set w.r.t. DPI''_{prev}. This holds due to

- the fact that DPI''_{prev} comprises a (not necessarily proper) subset of the test cases in DPI,

- Proposition 12.7 which states that a de-facto non-redundant node w.r.t. DPI cannot be pruned or replaced during any execution of DYNAMICHS with a current DPI that includes a (not necessarily proper) subset of the test cases in DPI and

- Lemma 12.10 which says that nd'_{suc} would be replaced or pruned in case that PRUNE is called given a witness of redundancy of nd'_{suc}.

So, we have derived a contradiction to the assumption that PRUNE is called given a minimal conflict set $X := \mathcal{C}''$ w.r.t. DPI''_{prev} which is a witness of redundancy of nd'_{suc}. Hence, case (s2) must be *true*.

Case (s2): If node is not the same node as nd, then node may or may not be de-facto non-redundant w.r.t. DPI. In the former case, the same argumentation as in case (s1) applies and yields a contradiction. In the latter case, we know that $\mathcal{C}'' \subset \mathsf{nd}'_{suc}.\mathsf{cs}[j]$ as well as $\mathsf{nd}'_{suc}[j] \in \mathsf{nd}'_{suc}.\mathsf{cs}[j] \setminus \mathcal{C}''$ must be *true* for some $j \in \{1, \ldots, |\mathsf{nd}'_{suc}| - 1\}$. So, by Lemma 12.10, nd'_{suc} is not an element of the returned list $\mathbf{Q}'$ of the call to PRUNE given the arguments $\mathbf{Q}$ (which includes nd'_{suc}), $X := \mathcal{C}''$ and $\mathbf{Q}_{dup}$.

However, at least one replacement node of nd'_{suc} must be found by PRUNE. This must hold by the following reasoning:

First, $\mathsf{nd} \in \mathbf{Q}_{dup}$ must hold at the time this call to PRUNE is made. This is satisfied since

- the entire (current) list $\mathbf{Q}_{dup}$ is browsed for an alternative subnode of nd'_{suc},

- $\mathsf{nd} \in \mathbf{Q}_{dup}$ holds at some point in time during the execution of DYNAMICHS with the current DPI DPI_{prev} due to the fact that node is not the same node as nd and the argumentation at the beginning of this proof,

- DPI_{prev} includes a subset of the test cases in DPI''_{prev},

- DPI''_{prev} includes a subset of the test cases in DPI,

- Proposition 12.7 states that a de-facto non-redundant node w.r.t. DPI cannot be pruned or replaced during the execution of DYNAMICHS with a current DPI that includes a subset of the test cases in DPI,

- nodes can only be deleted from $\mathbf{Q}_{dup}$ by being pruned and

- nd is de-facto non-redundant w.r.t. DPI.

Second, by line 21 and PRUNEQDUP, which are the only places in DYNAMICHS where $\mathbf{Q}_{dup}$ is modified, $\mathbf{Q}_{dup}$ is sorted in ascending order by node cardinality at any time during the execution of any call to DYNAMICHS.

Third, in order to construct a replacement node of nd'_{suc}, PRUNE first determines the maximal k such that $\mathcal{C}'' \subset \mathsf{nd}'_{suc}.\mathsf{cs}[k]$ and $\mathsf{nd}'_{suc}[k] \in \mathsf{nd}'_{suc}.\mathsf{cs}[k] \setminus \mathcal{C}''$. As case (c1) was proven to be false, we conclude that $k \leq |\mathsf{nd}'_{suc}| - 1$ must hold. Then, in line 96, an alternative subnode of nd'_{suc}

- which has cardinality $k + z$ where $z \geq 0$ is minimal and

- from which a replacement node of nd'_{suc} can be constructed

is searched for in $\mathbf{Q}_{dup}$. To see this, observe that elements in $\mathbf{Q}_{dup}$ – which is sorted in ascending order of node cardinality, as argued – are visited in order starting from the lowest cardinality node (line 96).

Fourth, $\mathsf{nd} \in \mathbf{Q}_{dup}$ is an alternative equal node of node. Since $\mathsf{nd}'_{suc} = \mathsf{node}_x$, we have that nd is an alternative subnode of nd'_{suc} such that $k \leq |\mathsf{nd}'_{suc}| - 1 = |\mathsf{nd}|$.

Thus, we have that one replacement node of nd'_{suc} is definitely found by PRUNE. And, in case there is only one replacement node of nd'_{suc} constructable during PRUNE, then this replacement node is given by $\mathsf{nd}'_{suc,new} := \mathrm{ADD}(\mathsf{nd}, x)$ with $\mathsf{nd}'_{suc,new}.\mathsf{cs} := \mathrm{ADD}(\mathsf{nd}.\mathsf{cs}, L)$. By the de-facto non-redundancy of nd and since x is specified as an element of the active sublabel of $\mathsf{nd}'_{suc}.\mathsf{cs}[|\mathsf{nd}'_{suc}|]$ w.r.t. DPI (see above), we obtain by Definition 12.7 that $\mathsf{nd}'_{suc,new}$ is a de-facto non-redundant node w.r.t. DPI. Thence, proposition (5) is *true*.

Due to $|\mathsf{nd}| = |\mathsf{node}| = |\mathsf{nd}'_{suc}| - 1$, the alternative subnode of nd'_{suc} *actually* found by PRUNE cannot have a cardinality greater than $|\mathsf{nd}'_{suc}| - 1$. So, let nd_{alt} be the found alternative subnode of nd'_{suc}. Since $|\mathsf{nd}_{alt}| \leq |\mathsf{nd}'_{suc}| - 1$, we obtain that the replacement node $\mathsf{nd}'_{suc,new,1}$ of nd'_{suc} constructed from nd_{alt} must meet $\mathsf{nd}'_{suc,new,1}[|\mathsf{nd}'_{suc}|] = \mathsf{nd}'_{suc}[|\mathsf{nd}'_{suc}|] = x$ as well as $\mathsf{nd}'_{suc,new,1}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] = \mathsf{nd}'_{suc}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] = L$. That is, the first $|\mathsf{nd}| = |\mathsf{node}| = |\mathsf{nd}'_{suc}| - 1$ positions of $\mathsf{nd}'_{suc,new,1}$ as a set correspond to a node in a transitive replaces-relation with nd.

Therefore, the same line of argument as used for nd'_{suc} can be applied to any node $\mathsf{nd}'_{suc,rep}$ in a transitive replaces-relation with nd'_{suc}. That is, the following must be valid for any node $\mathsf{nd}'_{suc,rep}$ in a transitive replaces-relation with nd'_{suc}:

- $\mathsf{nd}'_{suc,rep}[|\mathsf{nd}'_{suc}|] = x$ and $\mathsf{nd}'_{suc,rep}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] = L$.

- If PRUNE is called given a witness of redundancy of $\mathsf{nd}'_{suc,rep}$, then some replacement node of $\mathsf{nd}'_{suc,rep}$ is found. And, if only one replacement node of $\mathsf{nd}'_{suc,rep}$ is constructable, then this replacement node is de-facto non-redundant w.r.t. DPI.

After once a replacement node of nd'_{suc} or of some node in a transitive replaces-relation with nd'_{suc} is found which is de-facto non-redundant w.r.t. DPI, this replacement node cannot be replaced or pruned by Proposition 12.7. Therefore, by Lemma 12.10, no witness of redundancy of this replacement node can exist w.r.t. any DPI including a (not necessarily proper) subset of the test cases in DPI. Thence, proposition (6) is *true*.

Case (q2): Here, we have that nd'_{suc} is not the same node as nd_{suc}. This must be valid as nd'_{suc} is defined as the node set-equal to nd_{suc} that is an element of **Q** *immediately* after nd_{suc} was generated and nd_{suc} is assumed to be added to $\mathbf{Q}_{dup}$ after being generated.

Now, independently of whether **(c1) or (c2)** occurs, the following holds: If PRUNE is called given a witness of redundancy $\mathcal{C}''$ of nd'_{suc} w.r.t. DPI''_{prev}, then a replacement node of nd'_{suc} is found. And, if only one replacement node of nd'_{suc} is constructable, then this replacement node is de-facto non-redundant w.r.t. DPI.

To understand why this must hold, first recall that nd_{suc} is a successor of node, i.e. $\mathsf{nd}_{suc}[1..|\mathsf{nd}_{suc}| - 1]$ is the same node as node. Furthermore, node is the node set-equal to nd that is processed. That is, node is either the same node as nd or it is in a transitive replaces-relation with nd. Then, the same two cases (s1) and (s2) can be distinguished as in case (q1)(c2) where (s1) leads to a contradiction. So, case (s2) must be *true*. That is, node is not the same node as nd. Hence, by the argumentation in case (q1)(c2)(s2), $\mathsf{nd} \in \mathbf{Q}_{dup}$ must hold during the execution of any call to DYNAMICHS with a current DPI that comprises a (not necessarily proper) superset of the test cases in DPI_{prev} – which is the current DPI at the time nd is generated – and a (not necessarily proper) subset of the test cases in DPI. In particular, this implies that $\mathsf{nd} \in \mathbf{Q}_{dup}$ at the time PRUNE is called given the witness of redundancy $\mathcal{C}''$ of nd'_{suc} w.r.t. DPI''_{prev} as an argument.

By assumption, nd_{suc} has been added to $\mathbf{Q}_{dup}$ after being generated. Now, suppose PRUNEQDUP is called given a witness of redundancy $\mathcal{C}'$ of $\mathsf{nd}_{suc} \in \mathbf{Q}_{dup}$ w.r.t. some DPI DPI'_{prev} as an argument. Then DPI'_{prev} must comprise a (not necessarily proper) superset of the test cases in DPI_{prev}. This can be concluded from Lemma 12.12 which implies that nd_{suc} cannot have been generated during an execution of DYNAMICHS with a current DPI including a proper subset of the test cases in DPI_{prev}. Hence, the argumentation before implicates that $\mathsf{nd} \in \mathbf{Q}_{dup}$ at the time PRUNEQDUP is called given the witness of redundancy $\mathcal{C}'$ of nd_{suc} w.r.t. DPI'_{prev} as an argument.

Thus, nd_{suc} cannot be pruned on account of Lemma 12.8 which says that a node can only be pruned from $\mathbf{Q}_{dup}$ if the set $Comb_{\mathsf{nd}_{suc}}(\mathbf{Q}_{dup})$ of combined equal nodes of nd_{suc} of $\mathbf{Q}_{dup}$ (cf. Definition 12.5) is the empty set.

However, $Comb_{\mathsf{nd}_{suc}}(\mathbf{Q}_{dup}) \neq \emptyset$ must be valid. Because we demonstrated that

- $\mathsf{nd} \in \mathbf{Q}_{dup}$,

- $\mathsf{nd}_{suc} \in \mathbf{Q}_{dup}$,

- nd_{suc} is the same node as $\mathsf{node}_x = \mathrm{ADD}(\mathsf{node}, x)$ with $\mathsf{nd}_{suc}.\mathsf{cs}$ being equal to $\mathsf{node}_x.\mathsf{cs} = \mathrm{ADD}(\mathsf{node}.\mathsf{cs}, L)$,

- $\mathsf{nd} = \mathsf{node}$ and

- x is specified as an element of the active sublabel of $\mathsf{nd}_{suc}.\mathsf{cs}[|\mathsf{nd}_{suc}|]$ w.r.t. DPI (see above) wherefore $x \notin \mathsf{nd}_{suc}.\mathsf{cs}[|\mathsf{nd}_{suc}|] \setminus \mathcal{C}'$.

Therefore,

$$\mathsf{nd}_{comb} := \mathrm{ADD}(\mathsf{nd}, \mathsf{nd}_{suc}[|\mathsf{nd}| + 1..|\mathsf{nd}_{suc}|]) = \mathrm{ADD}(\mathsf{nd}, x)$$
$$\mathsf{nd}_{comb}.\mathsf{cs} := \mathrm{ADD}(\mathsf{nd}.\mathsf{cs}, \mathsf{nd}_{suc}.\mathsf{cs}[|\mathsf{nd}| + 1..|\mathsf{nd}_{suc}|]) = \mathrm{ADD}(\mathsf{nd}.\mathsf{cs}, L)$$

is a combined equal node of nd_{suc} of $\mathbf{Q}_{dup}$, i.e. $\mathsf{nd}_{comb} \in Comb_{\mathsf{nd}_{suc}}(\mathbf{Q}_{dup})$. The node nd_{comb} is de-facto non-redundant w.r.t. DPI as nd is de-facto non-redundant w.r.t. DPI and since x is an element of the active sublabel of $\mathsf{nd}_{suc}.\mathsf{cs}[|\mathsf{nd}_{suc}|]$ w.r.t. DPI.

By Definition 12.5, any combined equal node of nd_{suc} must share the element at the $|\mathsf{nd}_{suc}|$-th position with nd_{suc} and $\mathsf{nd}_{suc}.\mathsf{cs}$, respectively. Hence, the first $|\mathsf{nd}_{suc}| - 1$ elements of a combined equal node

of nd_{suc} are set-equal to the first $|\mathsf{nd}_{suc}| - 1$ elements of nd_{suc}. So, there exists a combined equal node, namely nd_{comb}, of any (redundant) node that results from nd_{suc} by a set of combined replacements.

By Lemma 12.11, the fact that $\mathsf{nd}_{comb} \in Comb_{\mathsf{nd}_{suc}}(\mathbf{Q}_{dup}) \subseteq Comb(\mathbf{Q}_{dup})$ at some point in time during the execution of DYNAMICHS with current DPI DPI'_{prev} and the de-facto non-redundancy of nd_{comb} w.r.t. DPI, we conclude that, during any execution of DYNAMICHS with a current DPI that includes a (not necessarily proper) superset of the test cases in DPI'_{prev} and includes a (not necessarily proper) subset of the test cases in DPI, $\mathsf{nd}_{comb} \in Comb(\mathbf{Q}_{dup})$ must hold. Because DPI'_{prev} is an arbitrary DPI that comprises a (not necessarily proper) superset of the test cases in DPI_{prev}, we derive that $\mathsf{nd}_{comb} \in Comb(\mathbf{Q}_{dup})$ must be $true$ particularly during the execution of DYNAMICHS with the current DPI DPI''_{prev}.

If $\mathcal{C}''$ is a witness of redundancy of $\mathsf{nd}_{suc} \in \mathbf{Q}_{dup}$, then the updated list $\mathbf{Q}_{dup}$ returned by PRUNEQDUP must include a combined replacement node of nd_{suc}, either nd_{comb} or some other node. Otherwise, i.e. if $\mathcal{C}''$ is not a witness of redundancy of $\mathsf{nd}_{suc} \in \mathbf{Q}_{dup}$, the updated list $\mathbf{Q}_{dup}$ returned by PRUNEQDUP must include nd_{suc}.

PRUNE is always called immediately after PRUNEQDUP and thus uses the updated list $\mathbf{Q}_{dup}$ which comprises a node set-equal to nd_{suc} and thus set-equal to nd'_{suc}. Consequently, we have that one replacement node of nd'_{suc} is definitely found by PRUNE. And, in case there is only one replacement node of nd'_{suc} constructable during PRUNE, this replacement node is given by nd_{comb}. Thence, proposition (5) is $true$.

Independently of which replacement node of nd'_{suc} is $actually$ found by PRUNE, a set-equality between this replacement node and nd_{comb} will hold. This is $true$ since $\mathsf{nd}_{comb} = \mathsf{nd}'_{suc}$ and since each replacement node, by definition, is set-equal to the node it replaces. Consequently, this set-equality holds for any node in a transitive replaces-relation with nd'_{suc}. So, we have that one replacement node of any node $\mathsf{nd}'_{suc,rep}$ in a transitive replaces-relation with nd'_{suc} is definitely found by PRUNE. And, in case there is only one replacement node of $\mathsf{nd}'_{suc,rep}$ constructable during PRUNE, this replacement node is given by nd_{comb} which is de-facto non-redundant w.r.t. DPI.

That nd_{comb}, after it has been used as a replacement node of nd'_{suc} or of some node in a transitive replaces-relation with nd'_{suc}, cannot be pruned or replaced, follows from Proposition 12.7 and the fact that nd_{comb} is de-facto non-redundant w.r.t. DPI. Therefore, by Lemma 12.10, no witness of redundancy of nd_{comb} can exist w.r.t. any DPI including a (not necessarily proper) subset of the test cases in DPI. Thence, proposition (6) is $true$. $\square$

The next result, Lemma 12.14, assumes an arbitrary fixed "current" DPI DPI such that DYNAMICHS with this "current" DPI DPI returns due to $\mathbf{Q} = [\,]$. Further on, it assumes an arbitrary minimal diagnosis $\mathcal{D}$ w.r.t. DPI and a node nd which is a proper subset of $\mathcal{D}$ such that nd is an element of $\mathbf{Q}$ anytime throughout all executions of DYNAMICHS during the execution of Algorithm 5 up to the one with the current DPI DPI. Additionally, nd cannot be pruned. It might be replaced; and in case there is only one potential replacement node of nd constructable from (the combined nodes of) $\mathbf{Q}_{dup}$, this replacement node is de-facto non-redundant w.r.t. DPI. Any node nd' in a transitive replaces relation with nd cannot be pruned either. It might again be replaced. In case there is only one potential replacement node of nd' constructable from (the combined nodes of) $\mathbf{Q}_{dup}$, this replacement node is de-facto non-redundant w.r.t. DPI.

Given these preconditions, the lemma establishes the existence of a node nd_{suc} that corresponds to a superset of nd and to a subset of $\mathcal{D}$, includes one element more than the set nd and is generated anytime throughout all executions of DYNAMICHS during the execution of Algorithm 5 up to the one with the current DPI DPI. Moreover, it states that the node nd'_{suc} set-equal to this generated node that is an element of $\mathbf{Q}$ cannot be pruned. However, it might be replaced. In case there is only one potential replacement node of nd'_{suc} constructable from (the combined nodes of) $\mathbf{Q}_{dup}$, this replacement node is de-facto non-redundant w.r.t. DPI. Any node $\mathsf{nd}'_{suc,rep}$ in a transitive replaces relation with nd'_{suc}

cannot be pruned either. It might again be replaced. In case there is only one potential replacement node of $\mathsf{nd}'_{suc,rep}$ constructable from (the combined nodes of) $\mathbf{Q}_{dup}$, this replacement node is de-facto non-redundant w.r.t. DPI.

Pictured, with respect to the hitting set tree constructed by DYNAMICHS, this lemma purports the following: Let the hitting set tree produced by DYNAMICHS be completely constructed for an arbitrary DPI DPI. In case there is any tree branch whose edge labels correspond to a part of the minimal diagnosis $\mathcal{D}$ w.r.t. DPI and which is known to be definitely not pruned during this tree construction, then this branch must be extended by one edge labeled by an element of $\mathcal{D}$ and this extended path is known to be definitely not pruned during this tree construction.

Lemma 12.14. *Assume the execution of* DYNAMICHS *with the current DPI DPI and assume that the execution stops due to $\mathbf{Q} = [\,]$. Let*

- *$GenNodes$ be the set of all nodes generated throughout the execution of all calls to* DYNAMICHS *during the execution of Algorithm 5,*

- *$\mathcal{D}$ be some minimal diagnosis w.r.t. DPI,*

- *DPI'_{prev} be a DPI which is either equal to DPI or includes fewer test cases than DPI and which is the current DPI during any particular call to* DYNAMICHS,

- nd *be some node such that the following holds:*

   - $\mathsf{nd} \subset \mathcal{D}$.
   - *There is some execution of* DYNAMICHS *with current DPI DPI'_{prev} during which it holds at some point in time that* $\mathsf{nd} \in \mathbf{Q}$.
   - *If* PRUNE *is called given a witness of redundancy of* nd, *then some replacement node of* nd *is found. If only one replacement node of* nd *is found, then this replacement node is de-facto non-redundant w.r.t. DPI.*
   - *Let* nd' *be in a transitive replaces-relation with* nd. *If* PRUNE *is called given a witness of redundancy of* nd', *then some replacement node of* nd' *is found. If only one replacement node of* nd' *is found, then this replacement node is de-facto non-redundant w.r.t. DPI.*

Then there are nodes nd_{suc} and nd'_{suc} such that the following holds:

(1) $\mathsf{nd} \subset \mathsf{nd}_{suc} \subseteq \mathcal{D}$.

(2) $|\mathsf{nd}_{suc}| = |\mathsf{nd}| + 1$.

(3) $\mathsf{nd}_{suc} \in GenNodes$.

(4) $\mathsf{nd}'_{suc} = \mathsf{nd}_{suc}$ is an element of $\mathbf{Q}$ immediately after nd_{suc} has been generated.

(5) If PRUNE *is called given a witness of redundancy of* nd'_{suc}, *then some replacement node of* nd'_{suc} *is found. If only one replacement node of* nd'_{suc} *is found, then this replacement node is de-facto non-redundant w.r.t. DPI.*

(6) Let $\mathsf{nd}'_{suc,rep}$ *be in a transitive replaces-relation with* nd'_{suc}. *If* PRUNE *is called given a witness of redundancy of* $\mathsf{nd}'_{suc,rep}$, *then some replacement node of* $\mathsf{nd}'_{suc,rep}$ *is found. If only one replacement node of* $\mathsf{nd}'_{suc,rep}$ *is found, then this replacement node is de-facto non-redundant w.r.t. DPI.*

Proof. Since nd $\in$ **Q** holds at some point in time during the execution of some call to DYNAMICHS with current DPI DPI'_{prev} and since the execution of DYNAMICHS with DPI terminates due to **Q** $= []$, we have that some node set-equal to nd must be processed. This must be satisfied because nodes can only be deleted from **Q** in that they are processed or pruned, and nd cannot be pruned from **Q**. For, by precondition, if PRUNE is called given a witness of redundancy of nd, then a replacement node of nd is found. And, if only one replacement node nd_{rep} of nd is found, nd_{rep} is de-facto non-redundant w.r.t. DPI.

Now, let nd_1 be a replacement node of nd found by PRUNE called with some witness of redundancy of nd. Then, by precondition, what holds for nd also holds for nd_1. That is, if PRUNE is called given a witness of redundancy of nd_1, then a replacement node of nd_1 is found. And, if only one replacement node $\mathsf{nd}_{1,rep}$ of nd_1 is found, $\mathsf{nd}_{1,rep}$ is de-facto non-redundant w.r.t. DPI.

The same holds for *any* nd_i which is in a transitive replaces-relation with nd. So, anytime PRUNE is called for a node set-equal to nd, at least one replacement node is found by PRUNE. And, in case nd_i is de-facto non-redundant w.r.t. DPI – which must be the case sooner or later for some node in a transitive replaces-relation with nd, by the given preconditions – then, by Proposition 12.7, nd_i cannot be pruned or replaced.

Hence, let us denote by node the node set-equal to nd that is finally processed. Let DPI_{prev} now be the "current" DPI of the execution of DYNAMICHS during which node is processed. Further, we denote the DPI of the immediate subsequent execution of DYNAMICHS by DPI_{prev+1}, and so on.

Since node is processed, it is either

- (s) labeled by a set (DLABEL returns in line 40, 46 or 34) or

- ($\neg$s) not labeled by a set (DLABEL returns in line 29 or 43).

Case ($\neg$s):   In this case, DLABEL returns

- (i) *nonmin* or

- (ii) *valid*.

Case (i):   By Lemma 12.1, node must be a non-minimal diagnosis w.r.t. DPI_{prev}. By line 15, node is then added to the set $\mathbf{D}_{\supset}$. $\mathbf{D}_{\supset}$ is never modified throughout Algorithm 5 and is given as an input argument to each subsequent call to DYNAMICHS by line 10 in Algorithm 5. During the execution of some subsequent call to DYNAMICHS using the DPI DPI_{prev+i} for $i \geq 1$, the set $\mathbf{D}_{\supset}$ might be modified by the PRUNE function called during UPDATETREE (line 65 and lines 70-78) or during DLABEL (line 38).

Recall that node is either the same node as nd or in a transitive replaces-relation with nd. Hence, by the argumentation given before, we have that, if PRUNE is called given a witness of redundancy of node, then there is a replacement node of node found by PRUNE. And, if there is only one replacement node of node found by PRUNE, then this replacement node is de-facto non-redundant w.r.t. DPI.

Therefore, node $\in \mathbf{D}_{\supset}$ cannot be pruned, i.e. node considered as a set cannot be deleted from $\mathbf{D}_{\supset}$ in line 65 or line 38. So, after any number of calls to PRUNE, we have that either node $\in \mathbf{D}_{\supset}$ or, otherwise, there is some node in $\mathbf{D}_{\supset}$ which is set-equal to node and which is in a transitive replaces-relation with node. We keep calling this (possibly replacement) node node in the following.

By Lemma 12.1, at the time node was processed, there must be some diagnosis $\mathcal{D}'$ w.r.t. DPI_{prev} such that $\mathcal{D}' \in \mathbf{D}_{calc}$ and node $\supset \mathcal{D}'$. Additionally, by Lemma 12.1, the set $\mathbf{D}_{calc}$ computed during DYNAMICHS for some "current" DPI DPI_j comprises only diagnoses w.r.t. DPI_j. Now, we have node $\subset \mathcal{D}$ since nd $\subset \mathcal{D}$ and node $=$ nd, and $\mathcal{D}' \subset$ node. That is, $\mathcal{D}' \subset \mathcal{D}$. By the precondition that $\mathcal{D}$ is a minimal diagnosis w.r.t. DPI, $\mathcal{D}'$ cannot be a diagnosis w.r.t. DPI. Thus, there cannot be any such $\mathcal{D}'$ in $\mathbf{D}_{calc}$ computed during DYNAMICHS for DPI.

All nodes in $\mathbf{D}_{calc}$ returned by some call to DYNAMICHS using DPI DPI_1 that are no diagnoses w.r.t. DPI_2, the extension of DPI_1 by a new query added as a positive or negative test case, are added to the set $\mathbf{D}_\times$ (and not to $\mathbf{D}_\checkmark$) in line 22 of Algorithm 5 and are thus no elements of the set $\mathbf{D}_\checkmark$ given as an argument to DYNAMICHS at the next call to DYNAMICHS. The elements of $\mathbf{D}_\checkmark$ given as an argument to DYNAMICHS at the next call to DYNAMICHS using DPI_2 are definitely added to $\mathbf{Q}$ again in lines 79-80 as $\mathbf{D}_\checkmark$ is not modified elsewhere in DYNAMICHS before lines 79-80 are reached.

Therefore, we need to differentiate between two cases: Either

- (x1) $\mathcal{D}' \in \mathbf{D}_\times$ never holds for the input argument $\mathbf{D}_\times$ to any call to DYNAMICHS or

- (x2) $\mathcal{D}' \in \mathbf{D}_\times$ holds at least once for the input argument $\mathbf{D}_\times$ to some call to DYNAMICHS.

Case (x1): Since $\mathcal{D}' \in \mathbf{D}_{calc}$ holds after the execution of DYNAMICHS using DPI_{prev} stops, we have that $\mathcal{D}' \in \mathbf{D}_\checkmark$ must hold for the argument $\mathbf{D}_\checkmark$ given to DYNAMICHS using DPI_{prev+1}. After UPDATETREE returns during DYNAMICHS using DPI_{prev+1}, $\mathcal{D}' \in \mathbf{Q}$ holds as argued. Subsequently, $\mathcal{D}'$ might be added again to $\mathbf{D}_{calc}$ and then to $\mathbf{D}_\checkmark$ again in line 21 of Algorithm 5 and to $\mathbf{Q}$ again in line 80 during DYNAMICHS using DPI_{prev+2}, and so forth. But, when a test case is added to some DPI DPI_{prev+i} in Algorithm 5 that invalidates the diagnosis $\mathcal{D}'$ (yielding the DPI $DPI_{prev+i+1}$), $\mathcal{D}' \notin \mathbf{D}_{calc}$ is assumed to hold (otherwise it would be an element of $\mathbf{D}_\times$ against our assumption). Such a test case must be added sometime as argued above. By Proposition 12.3, $\mathcal{D}'$ cannot be a (minimal) diagnosis w.r.t. any DPI including more test cases than $DPI_{prev+i+1}$ either. Notice that the case $\mathcal{D}' \notin \mathbf{D}_{calc}$ can emerge in spite of the fact that $\mathcal{D}'$ is a minimal diagnosis w.r.t. DPI_{prev+i} because there may be minimal diagnoses w.r.t. DPI_{prev+i} that have a higher probability than $\mathcal{D}'$. For $DPI_{prev+i+1}$ and all DPIs including more test cases than $DPI_{prev+i+1}$, $\mathcal{D}'$ cannot be added to $\mathbf{D}_{calc}$ anymore due to Lemma 12.1 which claims that only diagnoses w.r.t. the currently used DPI can be added to $\mathbf{D}_{calc}$.

Case (x2): Here, $\mathcal{D}' \in \mathbf{D}_\times$ holds at least once for the input argument $\mathbf{D}_\times$ to some call to DYNAMICHS using the DPI DPI_{prev+i}. Then, DYNAMICHS using the DPI $DPI_{prev+i-1}$ must have returned a set $\mathbf{D}_{calc}$ including $\mathcal{D}'$ as otherwise $\mathcal{D}'$ cannot be added to $\mathbf{D}_\times$. Hence, $\mathcal{D}'$ must be a diagnosis w.r.t. $DPI_{prev+i-1}$ by Lemma 12.1. Since $\mathcal{D}'$ is added to $\mathbf{D}_\times$, it cannot be a diagnosis w.r.t. DPI_{prev+i}. This must hold

- by Remark 7.4,

- since the set added to $\mathbf{D}_\times$ in Algorithm 5 is exactly the set $\mathbf{D}_{out}$ returned by GETINVALIDDIAGS in line 19 of Algorithm 5 and

- $\mathbf{D}_{out} = \mathbf{D}^+(Q)$ in case the user answer $u(Q)$ to the query Q w.r.t. $\mathbf{D}_{calc}$ and $DPI_{prev+i-1}$ is *false* and $\mathbf{D}_{out} = \mathbf{D}^-(Q)$ otherwise (notice that $\mathbf{D}_{calc}$ is referred to as $\mathbf{D}_\checkmark$ in Algorithm 5).

So, by Proposition 12.3, $\mathcal{D}'$ cannot be a (minimal) diagnosis w.r.t. any DPI including more test cases than DPI_{prev+i} either.

Each element in $\mathbf{D}_\times$ is processed by the UPDATETREE function (lines 48-69) called for the DPI DPI_{prev+i}. In lines 48-69, each node ndx in $\mathbf{D}_\times$ can only be pruned or either ndx or a node in a transitive replaces-relation with ndx is added to $\mathbf{Q}$ in line 68. $\mathbf{D}_{calc}$ is not modified by UPDATETREE and $\mathbf{D}_{calc} = \emptyset$ holds at the beginning of the execution of each call to DYNAMICHS. (A node set-equal to) $\mathcal{D}'$ cannot ever be readded to $\mathbf{D}_{calc}$ by Lemma 12.1 and since $\mathcal{D}'$ is not a diagnosis w.r.t any DPI including more test cases than DPI_{prev+i}. Hence, $\mathcal{D}' \in \mathbf{D}_{calc}$ can never hold for any DPI including more test cases than DPI_{prev+i}.

Hence, there must be some DPI DPI_{prev+k} such that $\mathbf{D}_\checkmark$ given as input to the DYNAMICHS-call for DPI_{prev+k} does not include *any* diagnosis $\mathcal{D}' \subset$ node. So, during the execution of the call to

DYNAMICHS using DPI DPI_{prev+k}, node must be deleted from $\mathbf{D}_\supset$ and be reinserted into $\mathbf{Q}$ by lines 70-78 in UPDATETREE which is called at the beginning of the execution of DYNAMICHS at any call to DYNAMICHS. This must hold since all nodes ndx in $\mathbf{D}_\supset$ that have not yet been pruned and for which there is no diagnosis in $\mathbf{D}_\checkmark$ which is a proper subset of ndx, are added to $\mathbf{Q}$ throughout lines 70-78. As shown, both criteria are met for node during the execution of the call to DYNAMICHS using DPI DPI_{prev+k}.

Case (ii): By Lemma 12.1, we know that node is a diagnosis w.r.t. DPI_{prev} and that node is added to $\mathbf{D}_{calc}$. Since node $\subset \mathcal{D}$ and $\mathcal{D}$ is a minimal diagnosis w.r.t. DPI, we obtain, by the same argumentation as in (i), that there must be some DPI DPI_{prev+k} such that $\mathbf{D}_\checkmark$ given as input to the DYNAMICHS-call for DPI_{prev+k} does not include node.

If node $\notin \mathbf{D}_\times$, then it cannot ever be added to $\mathbf{D}_{calc}$ again, as argued in case (i). Otherwise, during the execution of UPDATETREE which is called at the beginning of the execution of each call to DYNAMICHS, $\mathbf{D}_\times$ is modified in lines 48-69.

Now, we differentiate between two cases, namely node is either

- ($\neg$r) non-redundant w.r.t. DPI or

- (r) redundant w.r.t. DPI.

Case ($\neg$r): Due to the non-redundancy of node w.r.t. DPI, Lemma 12.4, Lemma 12.10 and Corollary 12.1, node cannot be replaced or pruned throughout lines 48-66. Thus, node is reinserted into $\mathbf{Q}$ in line 68.

Case (r): Since node is redundant w.r.t. DPI, it may or may not be redundant w.r.t. $DPI_{prev+k+1}$. So, during the UPDATETREE function called in DYNAMICHS for $DPI_{prev+k+1}$, there may or may not be some call to PRUNE given some X as argument which is a witness of redundancy of node. In the latter case, node will not be replaced or pruned during any PRUNE execution and will be reinserted into $\mathbf{Q}$ in line 68. In the former case, node might be replaced, but it cannot be pruned due to the same reasoning as given in the second paragraph of case (i). So, either node or some node in a transitive replaces-relation with node must be in $\mathbf{D}_\times$ at the time line 67 is reached. This node is then added to $\mathbf{Q}$ in line 68.

Now, both cases (i) and (ii) identified for case ($\neg$s) lead to the reinsertion of node or some node in a transitive replaces-relation with node – which is thus set-equal to nd – into $\mathbf{Q}$. Notice that this node has the same properties as node before one of the cases (i) or (ii) emerged (by analogue reasoning as conducted above). That is, if PRUNE is called given a witness of redundancy of node, then a replacement node of node is found. And, if only one replacement node of node is found, this replacement node is de-facto non-redundant w.r.t. DPI. So, we call this reinserted node again node.

Furthermore, node can be neither labeled by *valid* nor by *nonmin* during the execution of DYNAMICHS for DPI. This holds by Lemma 12.1 and since node can be neither a diagnosis nor a non-minimal diagnosis w.r.t. DPI due to node $\subset \mathcal{D}$ and the fact that $\mathcal{D}$ is a minimal diagnosis w.r.t. DPI. As a consequence of this and the assumption that the DYNAMICHS-call for DPI terminates due to $\mathbf{Q} = []$, case (s) must arise at some point in time for node during some execution of DYNAMICHS for some (previous) DPI not-necessarily equal to DPI.

Case (s): In this case, by Lemma 12.2, DLABEL returns a minimal conflict set L w.r.t. DPI_{prev} as a label for node where L has the property that $L \cap$ node $= \emptyset$.

It must hold that $L \neq \emptyset$. Otherwise, by Proposition 4.2, either

- (v1) $\mathcal{K}$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P_{prev}, N_{prev} \rangle_R$ where $DPI_{prev} = \langle \mathcal{K}, \mathcal{B}, P_{prev}, N_{prev} \rangle_R$ or

- (v2) DPI_{prev} is non-admissible.

In the former **case (v1)**, we know by Corollary 3.3 that the only (minimal) diagnosis w.r.t. DPI_{prev} is $\emptyset$. If DPI_{prev} is equal to DPI, this is a contradiction to the existence of some minimal diagnosis w.r.t. DPI, namely $\mathcal{D}$, which is not the empty set. $\mathcal{D} \supset \emptyset$ must hold since, by precondition, there is a node nd such that nd $\subset \mathcal{D}$ and since $\emptyset \subseteq$ nd.

Otherwise, if DPI_{prev} includes a proper subset of the test cases DPI includes, DPI can never be a current DPI during any execution of DYNAMICHS during the same execution of Algorithm 5 during which there is an execution of DYNAMICHS where DPI_{prev} is the current DPI. This holds as there must be at least two diagnoses in $\mathbf{D}_\checkmark$ in line 13 of Algorithm 5 in order for DYNAMICHS to be called again with a DPI including a proper superset of the test cases in DPI_{prev} (notice that, in Algorithm 5, the name of the set $\mathbf{D}_{calc}$ returned by DYNAMICHS for DPI_{prev} is $\mathbf{D}_\checkmark$). For, in case there is only one diagnosis, i.e. $\emptyset$, then the probability of this diagnosis is 1 which is greater or equal $1 - \sigma$ for any choice of σ due to $\sigma \geq 0$. Consequently, Algorithm 5 would return in line 14. This is a contradiction to the assumption that there is an execution of DYNAMICHS where DPI is the current DPI.

In the latter **case (v2)**, we can infer by Corollary 7.3, which states that adding queries as test cases to an admissible DPI can never yield a non-admissible DPI, that the DPI given as an input to Algorithm 5 must be non-admissible, contradiction.

Thence, $L \neq \emptyset$ and DYNAMICHS will execute lines 17-23 and generate one node $\text{node}_e := \text{ADD}(\text{node}, e)$ with $\text{node}_e.\text{cs} := \text{ADD}(\text{node.cs}, L)$ for each $e \in L$ (cf. Definition 12.2 for an explanation of the function ADD).

Now, we have that there must be some non-empty active sublabel of $L = \text{node}_e.\text{cs}[r]$ w.r.t. DPI where $r := |\text{node}_e|$ by Definition 12.6. Definition 12.6 is applicable by the following argumentation:

The first observation is that $\text{node}_e.\text{cs}[r]$ cannot be reduced twice during one and the same execution of DYNAMICHS using one and the same DPI DPI_{prev+j} which results from DPI_{prev} by addition of test cases. For, by Corollaries 12.1 and 12.2 and Lemmata 12.6 and 12.7, PRUNE as well as PRUNEQDUP can only be called given some minimal conflict set X w.r.t. DPI_{prev+j}. By Lemmata 12.10 and 12.8, all nodes ndx that are in the set returned by PRUNE and PRUNEQDUP, respectively, have the property that there are no proper supersets of X in ndx.cs. Moreover, there are no proper subsets of X in ndx.cs. Because each ndx.cs[m] for $m \in \{1, \ldots, |\text{ndx.cs}|\}$ must be a minimal conflict set w.r.t. some DPI equal to DPI_{prev+j} or including a subset of the test cases in DPI_{prev+j}. Otherwise, ndx could not be a node during the execution of DYNAMICHS where DPI_{prev+j} is the current DPI. By Proposition 12.1, there cannot be any $m \in \{1, \ldots, |\text{ndx.cs}|\}$ such that ndx.cs[m] $\subset X$ as X is a minimal conflict set w.r.t. DPI_{prev+j}. As two minimal conflict sets w.r.t. DPI_{prev+j} can never be in a proper subset-relationship with one another, $L = \text{node}_e.\text{cs}[r]$ can be modified at most once by PRUNE or PRUNEQDUP for the DPI DPI_{prev+j}.

Second, by Proposition 12.1, each minimal conflict set w.r.t. DPI_{prev} is a conflict set w.r.t. any DPI DPI_{prev+j} that results from DPI_{prev} by addition of test cases; that is, in particular, w.r.t. DPI. So, there must be some minimal conflict set $\mathcal{C}_j$ w.r.t. each DPI_{prev+j} such that $\mathcal{C}_j \subseteq L$ and there cannot be any minimal conflict set w.r.t. DPI_{prev+j} that is a proper superset of L.

Third, we have that $L \neq \emptyset$, L is a minimal conflict set w.r.t. DPI_{prev}, and DPI_{prev+j} includes a superset of the test cases in DPI_{prev}. Thus, by Proposition 12.2, each minimal conflict set w.r.t. DPI_{prev+j} must be non-empty. In particular, Proposition 12.2 implies that all minimal conflict sets w.r.t. DPI that are subsets of L must be non-empty.

By these three observations, the criteria of Definition 12.6 can be applied to analyze the active subnode of $\text{node}_e.\text{cs}[r]$ w.r.t. DPI. That is, if $\mathcal{C}_1, \ldots, \mathcal{C}_n$ is the (arbitrary actual) chronological sequence of all sets X given as an argument to PRUNE and PRUNEQDUP during all executions of DYNAMICHS from the one with current DPI DPI_{prev} up to and including the one with current DPI DPI where

- $\text{node}_e.\text{cs}[r] \supset \mathcal{C}_1$,

- each $\mathcal{C}_i$ is a minimal conflict set w.r.t. DPI_i for $i \in \{1, \ldots, n\}$

- $\mathcal{C}_k \supset \mathcal{C}_{k+1}$ for $k \in \{1, \ldots, n-1\}$,

- DPI_j includes a proper subset of the test cases DPI_{j+1} includes for $j \in \{1, \ldots, n-1\}$,

- DPI_n is equal to DPI or includes a proper subset of the test cases DPI includes and

- DPI_{prev} includes a proper subset of the test cases DPI_1 includes,

then $\mathcal{C}_n$ is the active sublabel of $\mathsf{node}_e.\mathsf{cs}[r]$ w.r.t. DPI. However, as argued before, the minimal conflict set $\mathcal{C}_n$ w.r.t. DPI_n cannot be the empty set. As a consequence, we obtain that there must be a non-empty active sublabel of $\mathsf{node}_e.\mathsf{cs}[r]$ w.r.t. DPI.

By Propositions 12.1 and 12.2, there is a non-empty minimal conflict set $\mathcal{C}'$ w.r.t. DPI such that $\mathcal{C}' \subseteq \mathcal{C}_n$. Due to $\mathcal{C}_n \subset \cdots \subset \mathcal{C}_1 \subset \mathsf{node}_e.\mathsf{cs}[r] = L$ we conclude that $\mathcal{C}_n \subset L$. Therefore, $\emptyset \subset \mathcal{C}' \subset L$ holds.

By Proposition 4.6, each minimal diagnosis w.r.t. DPI is a minimal hitting set of all minimal conflict sets w.r.t. DPI. Thence, we have that $\mathcal{C}' \cap \mathcal{D} \neq \emptyset$. So, by $\mathcal{C}' \subset L$, we have that $\emptyset \subset \mathcal{C}' \cap \mathcal{D} \subseteq L \cap \mathcal{D} \subseteq L$. Consequently, we define $\mathsf{nd}_{suc} := \mathsf{node}_x = \mathrm{ADD}(\mathsf{node}, x)$ with $\mathsf{nd}_{suc}.\mathsf{cs} := \mathsf{node}_x.\mathsf{cs} = \mathrm{ADD}(\mathsf{node}.\mathsf{cs}, L)$ for some $x \in \mathcal{C}' \cap \mathcal{D} \subseteq L$. Then, $\mathsf{nd}_{suc} \subseteq \mathcal{D}$ because $\mathsf{node} \subset \mathcal{D}$ and $x \in \mathcal{D}$. It is clear from the inference so far that $\mathsf{nd} \subset \mathsf{nd}_{suc}$, $|\mathsf{nd}_{suc}| = |\mathsf{nd}| + 1$ and $\mathsf{nd}_{suc} \in GenNodes$. This shows the truth of propositions (1)-(3).

Proposition (4) must hold by lines 20-23.

Now we argue why propositions (5) and (6) must hold. Assume that $\mathsf{nd}'_{suc} \in \mathbf{Q}$ is redundant w.r.t. some DPI DPI''_{prev} which is equal to DPI or includes fewer test cases than DPI. Then, there must be some minimal conflict set $\mathcal{C}''$ w.r.t. DPI''_{prev} which is a witness of redundancy of nd'_{suc}. Suppose that PRUNE is called given $X := \mathcal{C}''$ as an argument.

Now, we have to distinguish two cases: Either

- (q1) nd_{suc} was added to $\mathbf{Q}$ after it was generated or

- (q2) nd_{suc} was added to $\mathbf{Q}_{dup}$ after it was generated

(there are no other possibilities, see lines 17-23).

For each of these two cases, there are two more cases to discriminate between:

- (c1) $\mathcal{C}'' \subset \mathsf{nd}'_{suc}.\mathsf{cs}[|\mathsf{nd}'_{suc}|]$ and $\mathsf{nd}'_{suc}[|\mathsf{nd}'_{suc}|] \in \mathsf{nd}'_{suc}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] \setminus \mathcal{C}''$ or

- (c2) $\mathcal{C}'' \subset \mathsf{nd}'_{suc}.\mathsf{cs}[j]$ and $\mathsf{nd}'_{suc}[j] \in \mathsf{nd}'_{suc}.\mathsf{cs}[j] \setminus \mathcal{C}''$ for some $j \in \{1, \ldots, |\mathsf{nd}'_{suc}| - 1\}$.

Case (q1): Here, we have that nd'_{suc} is the same node as nd_{suc} since nd_{suc} was added to $\mathbf{Q}$ after generation and no node replacement can have taken place because nd'_{suc} is defined as the node set-equal to nd_{suc} that is an element of $\mathbf{Q}$ *immediately* after nd_{suc} has been generated. And, only one node corresponding to one and the same set can be in $\mathbf{Q}$ at the same time.

Case (c1): We have that $\mathcal{C}''$ must be equal to some minimal conflict set $\mathcal{C}_j$ in the sequence $\mathcal{C}_1, \ldots, \mathcal{C}_n$. This must be *true* since, first, DPI''_{prev} is equal to DPI or includes a subset of the test cases in DPI and DPI_{prev} includes a proper subset of the test cases in DPI''_{prev}.

To understand why the latter must hold, recall that DPI_{prev} is the DPI of the call to DYNAMICHS where nd_{suc} was generated and the minimal conflict set L was computed. By assumption, however, there is some minimal conflict set w.r.t. DPI''_{prev}, namely $\mathcal{C}''$, such that $\mathcal{C}'' \subset \mathsf{nd}'_{suc}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] = L$. Hence, it cannot be *true* that both L and $\mathcal{C}''$ are minimal conflict sets w.r.t. the same DPI. Otherwise, we would have

a contradiction to the minimality of L. By Proposition 12.1, which states that minimal conflict sets cannot grow by the addition of new test cases to the DPI, we obtain the claimed fact that DPI_{prev} includes a proper subset of the test cases in DPI''_{prev}.

Second, the sequence $\mathcal{C}_1, \ldots, \mathcal{C}_n$ comprises *all* sets X given as an argument to PRUNE and PRUNEQDUP during *all* executions of DYNAMICHS from the one with current DPI DPI_{prev} up to and including the one with current DPI DPI where $L = \mathsf{nd}'_{suc}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] \supset \mathcal{C}_1 \supset \cdots \supset \mathcal{C}_n$ holds. Reason for this to be valid is the fact that nd'_{suc} is the same node as nd_{suc} in the currently considered case (q1).

Now, recall $\mathcal{C}'$ is a minimal conflict set w.r.t. DPI such that $x \in \mathcal{C}' \cap \mathcal{D} \subset L$. Further, by $\mathsf{nd}'_{suc} = \mathsf{node}_x$, we have that $\mathsf{nd}'_{suc}[|\mathsf{nd}'_{suc}|] = x$. Since $\mathcal{C}' \subseteq \mathcal{C}_n$, we have that $\mathcal{C}' \subseteq \mathcal{C}_j$ must hold due to $\mathcal{C}_n \subseteq \mathcal{C}_j$. Therefore, we can infer by $\mathcal{C}'' = \mathcal{C}_j$ that $\mathcal{C}' \subseteq \mathcal{C}''$ is *true*. Now, $x \in \mathcal{C}'$ implies that $x \in \mathcal{C}''$ wherefore $x \notin \mathsf{nd}'_{suc}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] \setminus \mathcal{C}''$. By $x = \mathsf{nd}'_{suc}[|\mathsf{nd}'_{suc}|]$, this is a contradiction to the assumption of case (c1). Hence, case (c2) must arise.

Case (c2): We have that $\mathsf{nd}'_{suc}[1..|\mathsf{nd}'_{suc}|-1]$ must be redundant w.r.t. DPI''_{prev}. The subnode $\mathsf{nd}'_{suc}[1..|\mathsf{nd}'_{suc}|-1]$ of nd'_{suc} is the same node as node by $\mathsf{nd}'_{suc} = \mathsf{node}_x$. So, suppose PRUNE is called with arguments $\mathbf{Q}$ (which includes nd'_{suc}), $X := \mathcal{C}''$ and $\mathbf{Q}_{dup}$ during the execution of DYNAMICHS with current DPI DPI''_{prev}.

Recall that node is the node set-equal to nd that is processed. That is, node is either the same node as nd or it is in a transitive replaces-relation with nd. Therefore, by the preconditions of this lemma, the following holds: If PRUNE is called given a witness of redundancy of node, then a replacement node of node is found. And, if only one replacement node node_{rep} of node is found, then node_{rep} is de-facto non-redundant w.r.t. DPI.

So, at the time PRUNE might be called given a witness of redundancy of node, $Comb(\mathbf{Q}_{dup})$ must include a (non-necessarily proper) alternative subnode $\mathsf{node}_{rep,sub}$ of node from which the de-facto non-redundant node node_{rep} w.r.t. DPI can be constructed as

$$\mathsf{node}_{rep} := \mathrm{ADD}(\mathsf{node}_{rep,sub}, \mathsf{node}[|\mathsf{node}_{rep,sub}| + 1..|\mathsf{node}|])$$
$$\mathsf{node}_{rep}.\mathsf{cs} := \mathrm{ADD}(\mathsf{node}_{rep,sub}.\mathsf{cs}, \mathsf{node}.\mathsf{cs}[|\mathsf{node}_{rep,sub}| + 1..|\mathsf{node}|])$$

This holds due to

- Corollary 12.7, which says that each call to PRUNEQDUP returns the list $\mathbf{Q}_{dup}$, a subset of $Comb(\mathbf{Q}_{dup})$,

- the fact that PRUNEQDUP is always called immediately before PRUNE is called and

- the fact that PRUNE searches for alternative subnodes for the construction of a replacement node of a redundant node exactly in the output set of PRUNEQDUP.

By Definition 12.7, this is implies that $\mathsf{node}_{rep,sub}$ must be de-facto non-redundant w.r.t. DPI as otherwise the de-facto non-redundancy w.r.t. DPI could not hold for node_{rep}.

Consequently, by Lemma 12.11, $\mathsf{node}_{rep,sub} \in Comb(\mathbf{Q}_{dup})$ must always be satisfied during *any* execution of DYNAMICHS using a DPI that is equal to DPI or includes a subset of the test cases in DPI. Hence, in particular, this must hold for the DPI DPI''_{prev}.

By line 21 and PRUNEQDUP, which are the only places in DYNAMICHS where $\mathbf{Q}_{dup}$ is modified, $\mathbf{Q}_{dup}$ is sorted in ascending order by node cardinality at any time during the execution of any call to DYNAMICHS.

In order to construct a replacement node of nd'_{suc}, PRUNE first determines the maximal k such that $\mathcal{C}'' \subset \mathsf{nd}'_{suc}.\mathsf{cs}[k]$ and $\mathsf{nd}'_{suc}[k] \in \mathsf{nd}'_{suc}.\mathsf{cs}[k] \setminus \mathcal{C}''$. As case (c1) was proven to be false, we conclude that $k \leq |\mathsf{nd}'_{suc}| - 1$ must hold. Due to the fact that $\mathsf{nd}'_{suc}[1..|\mathsf{nd}'_{suc}| - 1]$ is the same node as node, as reasoned above, and the fact that a de-facto non-redundant alternative equal node node_{rep} (see above)

of node can be constructed from $\mathsf{node}_{rep,sub} \in Comb(\mathbf{Q}_{dup})$, we obtain that $k \leq |\mathsf{node}_{rep,sub}|$. This holds because the truth of both $\mathsf{node.cs}[m] \supset \mathcal{C}''$ and $\mathsf{node}[m] \in \mathsf{node.cs}[m] \setminus \mathcal{C}''$ for some $m \in \{|\mathsf{node}_{rep,sub}| + 1, \ldots, |\mathsf{node}|\}$ would be a contradiction to the de-facto non-redundancy of node_{rep} w.r.t. DPI.

Then, in line 96, an alternative subnode of nd'_{suc}

- which has cardinality $k + z$ where $z \geq 0$ is minimal and

- from which a replacement node of nd'_{suc} can be constructed

is searched for in $\mathbf{Q}_{dup}$. To see this, observe that elements in $\mathbf{Q}_{dup}$ – which is sorted in ascending order of node cardinality, as argued – are visited in order starting from the lowest cardinality node (line 96).

However, there is an alternative subnode $\mathsf{node}_{rep,sub}$ of node such that $k \leq |\mathsf{node}_{rep,sub}| \leq |\mathsf{node}| = |\mathsf{nd}'_{suc}| - 1$ and $\mathsf{node}_{rep,sub}$ is an element of the argument $\mathbf{Q}_{dup}$ given to PRUNE, as shown above. As nd'_{suc} is the same node as node_x, node is a subnode of nd'_{suc}. Therefore, $\mathsf{node}_{rep,sub}$ is an alternative subnode of nd'_{suc}.

Thus, we have that one replacement node of nd'_{suc} is definitely found by PRUNE. And, in case there is only one replacement node of nd'_{suc} constructable during PRUNE, then this replacement node is given by $\mathsf{nd}'_{suc,new} := \text{ADD}(\mathsf{node}_{rep,sub}, \mathsf{node}_x[|\mathsf{node}_{rep,sub}| + 1..|\mathsf{node}_x|]) = \text{ADD}(\mathsf{node}_{rep}, x)$ with $\mathsf{nd}'_{suc,new}.\mathsf{cs} := \text{ADD}(\mathsf{node}_{rep,sub}.\mathsf{cs}, \mathsf{node}_x.\mathsf{cs}[|\mathsf{node}_{rep,sub}| + 1..|\mathsf{node}_x|]) = \text{ADD}(\mathsf{node}_{rep}.\mathsf{cs}, L)$. As it is straightforward from the deductions above, $\mathsf{nd}'_{suc,new}$ is de-facto non-redundant w.r.t. DPI. Thence, proposition (5) is *true*.

Due to $|\mathsf{node}_{rep,sub}| \leq |\mathsf{node}| = |\mathsf{nd}'_{suc}| - 1$, the alternative subnode of nd'_{suc} *actually* found by PRUNE cannot have a cardinality greater than $|\mathsf{nd}'_{suc}| - 1$. So, let nd_{alt} be the found alternative subnode of nd'_{suc}. Since $|\mathsf{nd}_{alt}| \leq |\mathsf{nd}'_{suc}| - 1$, we obtain that the replacement node $\mathsf{nd}'_{suc,new,1}$ of nd'_{suc} constructed from nd_{alt} must meet $\mathsf{nd}'_{suc,new,1}[|\mathsf{nd}'_{suc}|] = \mathsf{nd}'_{suc}[|\mathsf{nd}'_{suc}|] = x$ as well as $\mathsf{nd}'_{suc,new,1}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] = \mathsf{nd}'_{suc}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] = L$. That is, the first $|\mathsf{node}| = |\mathsf{nd}'_{suc}| - 1$ positions as a set correspond to a node in a transitive replaces-relation with nd.

Now, we have the following precondition of this lemma: Let nd' be in a transitive replaces-relation with nd. If PRUNE is called given a witness of redundancy of nd', then some replacement node of nd' is found. If only one replacement node of nd' is found, then this replacement node is de-facto non-redundant w.r.t. DPI.

Therefore, the same line of argument as used for nd'_{suc} can be applied to any node $\mathsf{nd}'_{suc,rep}$ in a transitive replaces-relation with nd'_{suc}. That is, the following must be valid for any node $\mathsf{nd}'_{suc,rep}$ in a transitive replaces-relation with nd'_{suc}:

- $\mathsf{nd}'_{suc,rep}[|\mathsf{nd}'_{suc}|] = x$ and $\mathsf{nd}'_{suc,rep}.\mathsf{cs}[|\mathsf{nd}'_{suc}|] = L$.

- If PRUNE is called given a witness of redundancy of $\mathsf{nd}'_{suc,rep}$, then some replacement node of $\mathsf{nd}'_{suc,rep}$ is found. And, if only one replacement node of $\mathsf{nd}'_{suc,rep}$ is constructable, then this replacement node is de-facto non-redundant w.r.t. DPI.

After once a replacement node of nd'_{suc} or of some node in a transitive replaces-relation with nd'_{suc} is found which is de-facto non-redundant w.r.t. DPI, this replacement node cannot be replaced or pruned by Proposition 12.7. Therefore, by Lemma 12.10, no witness of redundancy of this replacement node can exist w.r.t. any DPI including a (not necessarily proper) subset of the test cases in DPI. Thence, proposition (6) is *true*.

Case (q2):   Here, we have that nd'_{suc} is not the same node as nd_{suc}. This must be valid as nd'_{suc} is defined as the node set-equal to nd_{suc} that is an element of $\mathbf{Q}$ *immediately* after nd_{suc} was generated and nd_{suc} is assumed to be added to $\mathbf{Q}_{dup}$ after being generated.

Now, independently of whether **(c1) or (c2)** occurs, the following holds: If PRUNE is called given a witness of redundancy of nd'_{suc}, then a replacement node of nd'_{suc} is found. And, if only one replacement node of nd'_{suc} is constructable, then this replacement node is de-facto non-redundant w.r.t. DPI.

To understand why this must hold, first recall that nd_{suc} is a successor of node, i.e. $\mathsf{nd}_{suc}[1..|\mathsf{nd}_{suc}| - 1]$ is the same node as node. Furthermore, node is the node set-equal to nd that is processed. That is, node is either the same node as nd or it is in a transitive replaces-relation with nd.

Therefore, by the preconditions of this lemma, the following holds: If PRUNE is called given a witness of redundancy of node, then a replacement node of node is found. And, if only one replacement node node_{rep} of node is constructable, then node_{rep} is de-facto non-redundant w.r.t. DPI.

As argued in case (q1)(c2), $Comb(\mathbf{Q}_{dup})$ must include a subnode $\mathsf{node}_{rep,sub}$ of node_{rep} that is de-facto non-redundant w.r.t. DPI and from which node_{rep} is constructed. This must be satisfied during any execution of DYNAMICHS using a DPI that is equal to DPI or includes a subset of the test cases in DPI. Hence, in particular, this must hold for the DPI DPI''_{prev}.

Since nd_{suc} has been added to $\mathbf{Q}_{dup}$ by assumption, it might be found to be redundant w.r.t. some DPI (either equal to DPI or including a subset of the test cases in DPI) during some execution of PRUNEQDUP. If so, nd_{suc} cannot be pruned on account of Lemma 12.8 which says that a node can only be pruned from $\mathbf{Q}_{dup}$ if the set $Comb_{\mathsf{nd}_{suc}}(\mathbf{Q}_{dup})$ of combined equal nodes of nd_{suc} of $\mathbf{Q}_{dup}$ (cf. Definition 12.5) is the empty set.

However, $Comb_{\mathsf{nd}_{suc}}(\mathbf{Q}_{dup}) \neq \emptyset$ must be valid. Because we demonstrated that

- $\mathsf{node}_{rep,sub} \in Comb(\mathbf{Q}_{dup})$,

- $\mathsf{nd}_{suc} \in \mathbf{Q}_{dup}$,

- nd_{suc} is the same node as $\mathsf{node}_x = \text{ADD}(\mathsf{node}, x)$ with $\mathsf{nd}_{suc}.\mathsf{cs}$ being equal to $\mathsf{node}_x.\mathsf{cs} = \text{ADD}(\mathsf{node}.\mathsf{cs}, L)$ and

- $x \notin \mathsf{nd}_{suc}.\mathsf{cs}[|\mathsf{nd}_{suc}|] \setminus \mathcal{C}''$ (see case (q1)(c1)) wherefore $\mathcal{C}''$ must be a witness of redundancy of node.

Therefore, $\mathsf{nd}_{comb} := \text{ADD}(\mathsf{node}_{rep,sub}, \mathsf{node}_x[|\mathsf{node}_{rep,sub}| + 1..|\mathsf{node}_x|]) = \text{ADD}(\mathsf{node}_{rep}, x)$ with $\mathsf{nd}_{comb}.\mathsf{cs} := \text{ADD}(\mathsf{node}_{rep,sub}.\mathsf{cs}, \mathsf{node}_x.\mathsf{cs}[|\mathsf{node}_{rep,sub}| + 1..|\mathsf{node}_x|]) = \text{ADD}(\mathsf{node}_{rep}.\mathsf{cs}, L)$ is a combined equal node of nd_{suc} of $\mathbf{Q}_{dup}$, i.e. $\mathsf{nd}_{comb} \in Comb_{\mathsf{nd}_{suc}}(\mathbf{Q}_{dup})$. As argued in case (q1)(c2), this node nd_{comb} (denoted by $\mathsf{nd}'_{suc,new}$ in case (q1)(c2)) is de-facto non-redundant w.r.t. DPI.

Because PRUNE is called immediately after PRUNEQDUP and thus uses the updated list $\mathbf{Q}_{dup}$ which comprises nd_{comb} and because $\mathsf{nd}_{comb} = \mathsf{nd}_{suc} = \mathsf{nd}'_{suc}$, we have that one replacement node of nd'_{suc} is definitely found by PRUNE. And, in case there is only one replacement node of nd'_{suc} constructable during PRUNE, this replacement node is given by nd_{comb}. Thence, proposition (5) is *true*.

By Proposition 12.7, the fact that $\mathsf{nd}_{comb} \in Comb_{\mathsf{nd}_{suc}}(\mathbf{Q}_{dup}) \subseteq Comb(\mathbf{Q}_{dup})$ at some point in time during the execution of DYNAMICHS with current DPI DPI''_{prev} and the de-facto non-redundancy of nd_{comb} w.r.t. DPI, we conclude that, during any execution of DYNAMICHS with a current DPI that includes a (not necessarily proper) superset of the test cases in DPI''_{prev} and includes a (not necessarily proper) subset of the test cases in DPI, $\mathsf{nd}_{comb} \in Comb(\mathbf{Q}_{dup})$ must hold. Further on, $\mathsf{nd}_{comb} = \mathsf{nd}'_{suc}$ is *true*.

Hence, independently of which replacement node of nd'_{suc} is *actually* found by PRUNE, a set-equality between this replacement node and nd_{comb} will hold. This is *true* since each replacement node, by definition, is set-equal to the node it replaces. Consequently, this set-equality holds for any node in a transitive replaces-relation with nd'_{suc}. So, we have that one replacement node of any node $\mathsf{nd}'_{suc,rep}$ in a transitive replaces-relation with nd'_{suc} is definitely found by PRUNE. And, in case there is only one replacement node of $\mathsf{nd}'_{suc,rep}$ constructable during PRUNE, this replacement node is given by nd_{comb} which is de-facto non-redundant w.r.t. DPI.

That nd_{comb}, after it has been used as a replacement node of nd'_{suc} or of some node in a transitive replaces-relation with nd'_{suc}, cannot be pruned or replaced, follows from Proposition 12.7 and the fact that nd_{comb} is de-facto non-redundant w.r.t. DPI. Therefore, by Lemma 12.10, no witness of redundancy of nd_{comb} can exist w.r.t. any DPI including a (not necessarily proper) subset of the test cases in DPI. Thence, proposition (6) is *true*.   □

In the following we prove the completeness of DYNAMICHS. Given an arbitrary minimal diagnosis $\mathcal{D}$ w.r.t. to an arbitrary fixed DPI DPI, Proposition 12.8 testifies that there must be some node set-equal to $\mathcal{D}$ that is processed during the execution of DYNAMICHS with current DPI DPI in case this execution terminates by reason of $\mathbf{Q} = []$. Second, the proposition demonstrates that the set $\mathbf{D}_{calc}$ returned by this execution of DYNAMICHS comprises all minimal diagnoses w.r.t. DPI. Additionally, the proposition shows that, at any point in time during the execution of Algorithm 5, some node that corresponds to a subset of $\mathcal{D}$ must be stored by DYNAMICHS.

In terms of the hitting set tree produced by DYNAMICHS, the proposition states that, after all branches in the tree have been closed or pruned, there is a closed branch labeled by $valid$ for each minimal diagnosis w.r.t. DPI. And, for any minimal diagnosis $\mathcal{D}$ w.r.t. DPI, at any time during the tree construction, there is some branch that corresponds to a part of $\mathcal{D}$.

This proposition will be proven by deriving the existence of a de-facto non-redundant node $\mathsf{nd}_{\mathcal{D}}$ w.r.t. DPI for any minimal diagnosis $\mathcal{D}$ w.r.t. DPI such that $\mathsf{nd}_{\mathcal{D}} \subseteq \mathcal{D}$. In case $\mathsf{nd}_{\mathcal{D}} = \mathcal{D}$, we will deduce directly that the proposition must be *true*. Otherwise, i.e. if $\mathsf{nd}_{\mathcal{D}} \subset \mathcal{D}$, then Lemmata 12.13 and 12.14 will be exploited.

Proposition 12.8 (Completeness of DYNAMICHS)**.** *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be the DPI and P' and N' the sets of positively and negatively answered queries given as an input to DYNAMICHS and assume that DYNAMICHS terminates due to $\mathbf{Q} = []$. Let further $DPI := \langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ and $\mathcal{D}$ be some minimal diagnosis w.r.t. DPI. Then the following holds:*

(1) At some point in time during the execution of DYNAMICHS with current DPI DPI, there is a node nd such that $\mathsf{nd} = \mathcal{D}$ and nd is processed.

(2) The execution of DYNAMICHS with current DPI DPI returns a set $\mathbf{D}_{calc}$ that comprises all minimal diagnoses w.r.t. DPI.

(3) Let DPI' be an arbitrary DPI that includes a (not necessarily proper) subset of the test cases in DPI. Then, at any point in time during the execution of DYNAMICHS with current DPI DPI', there is some node nd' such that $\mathsf{nd}' \subseteq \mathcal{D}$ and nd' is an element of one of the collections $\mathbf{Q}, \mathbf{D}_{calc}, \mathbf{D}_{\checkmark}, \mathbf{D}_{\times}$ or $\mathbf{D}_{\supset}$.

Proof. Let $GenNodes$ be the set of all nodes generated throughout the execution of all calls to DYNAMICHS during the execution of Algorithm 5.

Assume first that $\mathcal{D} = \emptyset$. This means that DPI must be the input DPI of Algorithm 5. Assume the opposite.

A query is only generated and added as a new test case to the DPI in lines 16 and 24 or 26 of Algorithm 5 if there are at least two diagnoses in the set $\mathbf{D}_{calc}$ (called $\mathbf{D}_{\checkmark}$ in Algorithm 5) returned by DYNAMICHS. Otherwise, line 16 cannot be reached since there must be exactly one diagnosis in $\mathbf{D}_{\checkmark}$ when it comes to the execution of line 13 wherefore the probability of this diagnosis must be equal to 1 which is greater or equal to $1 - \sigma$ for any choice of σ (recall that σ is positive). Please notice that $\mathbf{D}_{\checkmark} = \emptyset$ cannot hold in line 13 since this would imply the non-admissibility of the input DPI given to Algorithm 5 by Corollary 7.3 and Definition 3.6. By precondition, however, the DPI provided as an input to Algorithm 5 must be admissible.

Now, since DPI is assumed to be not equal to the input DPI of Algorithm 5, we have, by the argumentation given, that there must have been at least two diagnoses w.r.t. the input DPI.

Let us first assume that $\mathcal{K}$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$ where $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ is the input DPI. Then, by Corollary 3.3, $\emptyset$ is a diagnosis w.r.t. the input DPI. Obviously, it must be a minimal diagnosis and the only minimal diagnosis w.r.t. the input DPI, contradiction.

Second, suppose that $\mathcal{K}$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$. By Proposition 4.6 which says that a diagnosis w.r.t. some DPI is a hitting set of all minimal conflict sets w.r.t. this DPI, we conclude that there must be at least one minimal conflict set $\mathcal{C}$ w.r.t. the input DPI. Now, by Proposition 12.1, there must be a minimal conflict set $\mathcal{C}'$ w.r.t. DPI such that $\mathcal{C}' \subseteq \mathcal{C}$. By Proposition 4.2, the fact that $\mathcal{K}$ is invalid w.r.t. $\langle \cdot, \mathcal{B}, P, N \rangle_R$, the fact that the input DPI is admissible and Corollary 7.3 which states that the addition of queries as test cases cannot make an admissible DPI non-admissible, we obtain that $\emptyset \subset \mathcal{C}'$. By Proposition 4.6, this is a contradiction to $\mathcal{D} = \emptyset$ and the fact that $\mathcal{D}$ is a diagnosis w.r.t. DPI.

So, DPI is the input DPI. Hence, the first call to DYNAMICHS throughout the execution of Algorithm 5 considers this DPI. During the execution of the first call to DYNAMICHS, $\mathbf{Q} = [\emptyset]$ holds by lines 3 and 10 of Algorithm 5. The function UPDATETREE has no effect during the execution of the first call to DYNAMICHS in Algorithm 5. That is, in particular, it does not modify $\mathbf{Q}$. For, UPDATETREE first iterates over all elements in $\mathbf{D}_\times$, then over all elements in $\mathbf{D}_\supset$ and finally over all elements in $\mathbf{D}_\checkmark$ where $\mathbf{D}_\times = \mathbf{D}_\supset = \mathbf{D}_\checkmark = \emptyset$ by lines 1 and 10 in Algorithm 5. Hence, $\mathbf{Q} = [\emptyset]$ holds when DYNAMICHS reaches line 6 wherefore $\emptyset$ is processed.

Now, assume $\mathcal{D} \neq \emptyset$. In this case, the root node must be labeled by some minimal conflict set L w.r.t. the DPI given as input to Algorithm 5. To see this, suppose the opposite, i.e. that the root node is labeled by (i) $nonmin$ or (ii) $valid$.

Case (i): This leads to a contradiction. For, $\mathbf{D}_{calc} = \emptyset$ holds at the beginning of each execution of DYNAMICHS (line 3). The root node $\emptyset$ must be the first node that is processed throughout all executions of DYNAMICHS during the execution of Algorithm 5 since it holds for each other node node that node $\supset \emptyset$. Thus, the non-minimality criterion (lines 27-29) cannot be satisfied because $\mathbf{D}_{calc} = \emptyset$ must hold in line 27 when DLABEL is executed for the root node. Hence, the label $nonmin$ is impossible for the node $\emptyset$.

Case (ii): By Lemma 12.1, we can deduce that $\emptyset$ is a diagnosis w.r.t. the input DPI. The fact that there cannot be any diagnosis w.r.t. the input DPI which is a proper subset of $\emptyset$ implies that $\emptyset$ is a minimal diagnosis w.r.t. the input DPI. By the reasoning applied before (in the case $\mathcal{D} = \emptyset$), we obtain that DPI is equal to the input DPI and that $\emptyset$ is the only minimal diagnosis w.r.t. DPI. This is a contradiction to the existence of a minimal diagnosis w.r.t. DPI, namely $\mathcal{D}$, which is non-empty.

Consequently, the root node must be labeled by some minimal conflict set L w.r.t. the input DPI. Hence, DYNAMICHS will execute lines 17-23 and generate one node $node_e := \text{ADD}(\emptyset, e) = [e]$ with $node_e.\mathsf{cs} := \text{ADD}(\emptyset, L) = [L]$ for each $e \in L$ (cf. Definition 12.2 for an explanation of the function ADD). This means that $node_e \in GenNodes$ for each $e \in L$. As L is a set and thus comprises only one exemplar of each element, there cannot be a set-equal node $node_e'$ of $node_e$ in $\mathbf{Q}$ at the time $node_e$ is generated. So, each $node_e$ must be added to $\mathbf{Q}$ in line 23.

By Proposition 12.1, there must be some minimal conflict set $\mathcal{C}$ w.r.t. DPI such that $\mathcal{C} \subseteq L$. Since $\mathcal{D}$ is a diagnosis w.r.t. DPI, we have that $\mathcal{C} \cap \mathcal{D} \neq \emptyset$ by Proposition 4.6. Thence, $L \cap \mathcal{D} \neq \emptyset$ must be $true$. Therefore, in particular, $L \neq \emptyset$ must hold.

Assume that $|\mathcal{D}| = 1$. This implies by Proposition 4.6 that each minimal conflict set w.r.t. DPI includes x. Further, there is some $x \in L$ such that $\mathcal{D} = \{x\} = node_x$. By Corollary 12.1 and Lemmata 12.6 and 12.7, PRUNE is only called given some minimal conflict set X w.r.t. the current DPI DPI_{prev} as argument. As DYNAMICHS using DPI is assumed to terminate due to $\mathbf{Q} = []$, DPI_{prev} must be equal to DPI or include only a subset of the test cases DPI includes. By Proposition 12.1, it must hold for X that it is equal to or a superset of some minimal conflict set w.r.t. DPI. Hence $x \in X$ must hold wherefore X cannot be a witness of redundancy of $node_x$. So, $node_x$ can never be pruned and must be finally processed as DPI terminates due to $\mathbf{Q} = []$ and nodes can only be deleted from $\mathbf{Q}$ by being pruned or processed. So far, we have established the truth of the lemma for $|\mathcal{D}| \leq 1$.

Now, suppose $|\mathcal{D}| \geq 2$. In the following, we argue that there must be some node $\mathsf{node}_y \subset \mathcal{D}$ for some $y \in L$ which is de-facto non-redundant w.r.t. DPI.

As DYNAMICHS using DPI is assumed to terminate due to $\mathbf{Q} = []$, each node node_e for $e \in L$ must have been generated (and L must have been computed) during DYNAMICHS with some current DPI DPI_{prev} which is equal to DPI or includes only a subset of the test cases DPI includes. Let DPI_{prev+i} be any DPI which includes a proper superset of the test cases DPI_{prev} includes and is either equal to DPI or comprises a subset of the test cases DPI comprises. Then, Proposition 12.1 manifests that there must be some minimal conflict set $\mathcal{C}_i$ w.r.t. DPI_{prev+i} such that $\mathcal{C}_i \subseteq L$. Since we proved above that $L \neq \emptyset$ must hold, we deduce by Proposition 12.2 that $\mathcal{C}_i \neq \emptyset$ must be valid.

From Corollaries 12.1, 12.2 and Lemmata 12.6 and 12.7 we infer that PRUNE as well as PRUNEQDUP are always called with a minimal conflict set X w.r.t. the current DPI given as an argument. Lemma 12.8 and the fact that PRUNE is always called immediately after PRUNEQDUP given the argument $\mathbf{Q}_{dup}$ which is the output list of PRUNEQDUP, we have that the list $\mathbf{Q}_{dup}$ includes only nodes nd such that there is no $r \in \{1, \ldots, |\mathsf{nd}|\}$ for which $\mathsf{nd.cs}[r] \supset X$. As a consequence of this, we have by Lemma 12.10 that for all nodes nd in the collection S' returned by PRUNE there is no $r \in \{1, \ldots, |\mathsf{nd}|\}$ for which $\mathsf{nd.cs}[r] \supset X$.

Thence, the first time PRUNE is called with some $X_1 \subset L$, X_1 is a minimal conflict set w.r.t. some DPI DPI_{prev+i}. Thus, as argued, $X_1 \supset \emptyset$ must hold. So, after PRUNE has finished executing, for each node node in its output set there will be no $r \in \{1, \ldots, |\mathsf{node}|\}$ such that $\mathsf{node.cs}[r] \supset X_1$. For any further minimal conflict set X_2 w.r.t. some $DPI_{prev+i+k}$ for which PRUNE is called, we have that $X_2 \supset \emptyset$ and for each node node in its output set there will be no $r \in \{1, \ldots, |\mathsf{node}|\}$ such that $\mathsf{node.cs}[r] \supset X_2$, and so on.

For L, in particular, there is some (possibly empty) sequence of minimal conflict sets $X_1, \ldots, X_n$ w.r.t. DPIs $DPI_{prev+i_1}, \ldots, DPI_{prev+i_n}$ ($i_j < i_{j+1}$ for $j \in \{1, \ldots, n-1\}$) such that $L \supset X_1$ and $X_i \supset X_{i+1}$ for $i \in \{1, \ldots, n\}$ where this sequence includes all such conflict sets which restrict a conflict set used to label nodes that was initially given by L. Since X_n is a minimal conflict set w.r.t. DPI_{prev+i_n} which is equal to DPI or includes only a subset of the test cases DPI includes, we have that there must be some minimal conflict set $\mathcal{C}$ w.r.t. DPI such that $\mathcal{C} \subseteq X_n$, as already argued. As $\mathcal{D}$ must hit $\mathcal{C}$ by Proposition 4.6, we obtain that $\mathcal{D} \cap X_n \neq \emptyset$.

So, by the inference given, there must be some $y \in L$ such that $y \in X_1 \cap \cdots \cap X_n$ and $y \in \mathcal{D}$. That is, $\mathsf{node}_y \subset \mathcal{D}$.

Since $|\mathsf{node}_e| = 1$ and $\mathsf{node}_e.\mathsf{cs}[1] = L$ for all $e \in L$, in particular for $e = y$, we obtain by Definitions 12.6 and 12.7 that node_y is de-facto non-redundant w.r.t. DPI.

So, the preconditions of Lemma 12.13 are met for node_y. As a consequence, there must be a node nd'_{suc} such that $|\mathsf{nd}'_{suc}| = |\mathsf{node}_y| + 1$, $\mathsf{nd}'_{suc} \subseteq \mathcal{D}$, nd'_{suc} is an element of $\mathbf{Q}$ immediately after node_y has been processed and nd'_{suc} satisfies the postulations to the node nd in the preconditions of Lemma 12.14. Hence, if $\mathsf{nd}'_{suc} \subset \mathcal{D}$, there must be a node nd''_{suc} such that $|\mathsf{nd}''_{suc}| = |\mathsf{nd}'_{suc}| + 1$, $\mathsf{nd}''_{suc} \subseteq \mathcal{D}$, nd''_{suc} is an element of $\mathbf{Q}$ immediately after a node set-equal to nd'_{suc} has been processed and nd''_{suc} satisfies the postulations to the node nd in the preconditions of Lemma 12.14.

This reasoning by means of Lemma 12.14 can be further applied to finally derive that some node $\mathsf{nd} = \mathcal{D}$ must be generated and some node nd' set-equal to nd must be an element of $\mathbf{Q}$. By Lemma 12.14, either nd' or a node set-equal to nd' which is in a transitive replaces-relation with nd' must finally be processed. Reason for this is that $\mathsf{nd}' \in \mathbf{Q}$ cannot be pruned, but can only be replaced, and each replacement node is set-equal to nd' and thus to $\mathcal{D}$. Moreover, the execution of DYNAMICHS with current DPI DPI terminates due to $\mathbf{Q} = []$ wherefore each node in $\mathbf{Q}$ must be either pruned or processed as these are the only two ways nodes might be eliminated from $\mathbf{Q}$.

If some node $\mathsf{nd} = \mathcal{D}$ is processed during an execution of DYNAMICHS with current DPI some DPI DPI' that includes a proper subset of the test cases in DPI, then DLABEL cannot return a set L. This holds by Lemma 12.2 and Proposition 12.1. The former says that $\mathsf{nd} \cap L = \emptyset$ and L is a minimal conflict set w.r.t. DPI'. The latter asserts that each conflict set w.r.t. DPI' is a conflict set w.r.t. DPI. Moreover,

we can deduce that $L \neq \emptyset$ must hold if a set L is returned by DLABEL by a similar argumentation as used in the proof of Lemma 12.14. That is, by Proposition 4.6, we have that $\mathcal{D}$ cannot be a diagnosis w.r.t. DPI, contradiction.

Hence, DLABEL must return *nonmin* or *valid* for nd. In the former case, it would be added to $\mathbf{D}_{\supset}$, in the latter to $\mathbf{D}_{calc}$. Similarly as done in the proof of Lemma 12.14, we can show that nd must be reinserted into $\mathbf{Q}$ the latest during the execution of DYNAMICHS with current DPI DPI and, in particular, nd must be an element of $\mathbf{Q}$ when the repeat-loop during the execution of DYNAMICHS with current DPI DPI is entered. Thus, nd must be (again) processed during the execution of DYNAMICHS with current DPI DPI. This proves proposition (1).

Proposition (2): At the beginning of each execution of DYNAMICHS, it holds that $\mathbf{D}_{calc} = \emptyset$. This is *true* in particular for the execution of DYNAMICHS with current DPI DPI. Now, proposition (1) reveals that, for *each* diagnosis $\mathcal{D}$ w.r.t. DPI, at some point in time during the execution of DYNAMICHS with current DPI DPI, there is a node nd such that nd $= \mathcal{D}$ and nd is processed. When nd is processed, the DLABEL function is called for nd. The DLABEL function might return (a) a set L, (b) *nonmin* or (c) *valid*. There are no other possible return values of DLABEL.

Case (a): By Lemma 12.2, L must be a minimal conflict set w.r.t. DPI such that nd $\cap L = \emptyset$. According to Proposition 4.6, it must hold for $\mathcal{D}$ that $\mathcal{D} \cap L \neq \emptyset$ since $\mathcal{D}$ is a minimal diagnosis w.r.t. DPI. Since $\mathcal{D} = $ nd, we obtain a contradiction.

Case (b): By Lemma 12.1, $\mathbf{D}_{calc}$ can comprise only diagnoses w.r.t. DPI. By line 27, this yields that there is a diagnosis w.r.t. DPI that is a proper subset of nd. This however is a contradiction to the set-equality of nd with the minimal diagnosis $\mathcal{D}$ w.r.t. DPI.

Consequently, case (c) must arise. This implies that nd is added to $\mathbf{D}_{calc}$ in line 13.

Proposition (3) is a direct consequence of the reasoning in this proof and in the proofs of Lemmata 12.13 and 12.14. $\qquad\square$

12.4.9 Soundness of DYNAMICHS

Having established the completeness of each call to DYNAMICHS concerning the minimal diagnoses w.r.t. the current DPI DPI at this call, we are now able to prove the soundness of each call to DYNAMICHS. That is, we will demonstrate that *only* minimal diagnoses w.r.t. DPI can be added to the set $\mathbf{D}_{calc}$ during DYNAMICHS with the current DPI DPI. Necessary condition for the proof of the following proposition is the completeness of DYNAMICHS, i.e. Proposition 12.8.

Proposition 12.9 (Soundness of DYNAMICHS)**.** *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be the DPI and P' and N' the sets of positively and negatively answered queries given as an input to DYNAMICHS. Let further $DPI :=$ $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. Then, the following holds:*

(1) At any point in time during the execution of DYNAMICHS with current DPI DPI, each node in $\mathbf{D}_{calc}$ is a minimal diagnosis w.r.t. DPI.

(2) At any point in time during the execution of DYNAMICHS with current DPI DPI, $\mathbf{D}_{calc}$ comprises the $|\mathbf{D}_{calc}|$ most-probable minimal diagnoses w.r.t. DPI.

Proof. Proposition (1): At the beginning of any execution of DYNAMICHS, the set $\mathbf{D}_{calc}$ is the empty set (line 3). So, it suffices to show that only minimal diagnoses w.r.t. DPI can be added to $\mathbf{D}_{calc}$ during the execution of DYNAMICHS with the current DPI DPI.

A node node can be added to $\mathbf{D}_{calc}$ exclusively in line 13. In order for this line to be reached, by the criterion that is checked in line 12, node must be processed and labeled by *valid*. By Lemma 12.1, if node gets labeled by *valid*, then it is a diagnosis w.r.t. DPI.

So, assume that node is added to $\mathbf{D}_{calc}$ where node is a non-minimal diagnosis w.r.t. DPI. Since node must have been processed and labeled by *valid*, the DLABEL function must have been executed

given node as an argument and must have returned in line 43. Hence, there can be no node $\mathsf{nd} \in \mathbf{D}_{calc}$ such that $\mathsf{nd} \subset \mathsf{node}$ holds, as otherwise DLABEL would have already returned in line 29.

However, since node is a non-minimal diagnosis w.r.t. DPI there must be some minimal diagnosis $\mathcal{D}$ w.r.t. DPI such that $\mathcal{D} \subset \mathsf{node}$. Moreover, by Proposition 12.8, at any point in time before $\mathcal{D}$ is added to $\mathbf{D}_{calc}$, there must be some node nd such that $\mathsf{nd} \subseteq \mathcal{D}$ and nd is an element of one of the collections (a) $\mathbf{D}_{calc}$, (b) $\mathbf{D}_{\checkmark}$, (c) $\mathbf{D}_{\times}$, (d) $\mathbf{D}_{\supset}$ or (e) $\mathbf{Q}$. So, let us consider these cases in sequence.

Case (a): First, $\mathsf{nd} \subseteq \mathcal{D}$ and $\mathcal{D} \subset \mathsf{node}$ implies that $\mathsf{nd} \subset \mathsf{node}$ must be valid. As mentioned above, there can be no node in $\mathbf{D}_{calc}$ which is a proper subset of node, contradiction.

Case (b): In this case, nd must be also an element of $\mathbf{Q}$ since all nodes in $\mathbf{D}_{\checkmark}$ are inserted into $\mathbf{Q}$ during UPDATETREE which is executed before the repeat-loop is entered, i.e. before it can come to the assumed addition of node to $\mathbf{D}_{calc}$ which can only take place within the repeat-loop. So, in fact case (e) applies here.

Case (c): As can be easily seen from lines 67-69 in UPDATETREE, $\mathbf{D}_{\times}$ must be the empty set at the time node might be added to $\mathbf{D}_{calc}$ by analogue argumentation as in case (b), contradiction.

Case (d): By lines 70-78 in UPDATETREE and the fact that UPDATETREE must have been executed before the assumed addition of node to $\mathbf{D}_{calc}$ can take place as argued in case (b), we have that there must be some node $\mathsf{nd}_{sub} \in \mathbf{D}_{\checkmark}$ such that $\mathsf{nd}_{sub} \subset \mathsf{nd}$. Otherwise, nd would have been deleted from $\mathbf{D}_{\supset}$ in line 78. By $\mathsf{nd} \subset \mathsf{node}$ as per case (a), we deduce that $\mathsf{nd}_{sub} \subset \mathsf{node}$. Due to $\mathsf{nd} \subseteq \mathcal{D}$, it must be *true* that $\mathsf{nd}_{sub} \subseteq \mathcal{D}$. Thus, we have derived that case case (b) holds for the node nd_{sub}. By the deductions in case (b) above, we eventually know that case (e) must hold.

Thence, assumption of cases (a) and (c) is contradictory. Cases (b) and (d) imply the truth of case (e). Therefore, case (e) must occur.

Case (e): Due to the facts that all nodes are inserted into $\mathbf{Q}$ in a manner that descending order of nodes in $\mathbf{Q}$ by $p_{nodes}()$ is maintained (cf. lines 23, 100 and 103) and always the first node in $\mathbf{Q}$ is processed next (cf. line 6), we conclude that $p_{nodes}(\mathsf{nd}) \leq p_{nodes}(\mathsf{node})$ must be valid. However, due to $\mathsf{nd} \subseteq \mathcal{D} \subset \mathsf{node}$ we have that $\mathsf{nd} \subset \mathsf{node}$. Now, by Lemma 4.14, $p_{nodes}(\mathsf{n}) > p_{nodes}(\mathsf{n}')$ holds for any two nodes n and n' such that $\mathsf{n} \subset \mathsf{n}'$. Therefore, $p_{nodes}(\mathsf{nd}) > p_{nodes}(\mathsf{node})$, contradiction.

Proposition (2): By proposition (1), each node added to $\mathbf{D}_{calc}$ must be a minimal diagnosis w.r.t. DPI.

Assume any point in time t during the execution of DYNAMICHS with the current DPI DPI. Then, $|\mathbf{D}_{calc}| = m \geq 0$ must hold. We use induction by m to prove proposition (2).

Base Case: Suppose that $m = 0$ and some minimal diagnosis $\mathcal{D}$ w.r.t. DPI is added to $\mathbf{D}_{calc}$ where $\mathcal{D}$ is not the most probable minimal diagnosis w.r.t. DPI. This implies that $\mathcal{D}$ is processed and that $\mathcal{D}$ has the highest probability as per $p_{nodes}()$ among all nodes that are elements of $\mathbf{Q}$ at time t, as argued in the proof of proposition (1).

Let us denote by $\mathcal{D}_1$ the most probable minimal diagnosis w.r.t. DPI. That is, $p_{nodes}(\mathcal{D}_1) > p_{nodes}(\mathcal{D})$ holds.

Then, by Proposition 12.8, at any point in time during the execution of DYNAMICHS with the current DPI DPI, there must be some node nd_1 such that $\mathsf{nd}_1 \subseteq \mathcal{D}_1$ and nd_1 is an element of one of the collections (a) $\mathbf{D}_{calc}$, (b) $\mathbf{D}_{\checkmark}$, (c) $\mathbf{D}_{\times}$, (d) $\mathbf{D}_{\supset}$ or (e) $\mathbf{Q}$.

Case (a) can be ruled out due to the assumption that $\mathbf{D}_{calc} = \emptyset$. Cases (b)-(d) can be treated analogously as above in the proof of proposition (1). Hence, case (e) must hold.

That is, $\mathsf{nd}_1 \in \mathbf{Q}$ at time t and nd_1 is equal to or a subset of $\mathcal{D}_1$. As $p_{nodes}(\mathsf{nd}_1) \geq p_{nodes}(\mathcal{D}_1) > p_{nodes}(\mathcal{D})$ holds by Lemma 4.14, we can infer that $\mathcal{D}$ has not the highest probability as per $p_{nodes}()$ among all nodes that are elements of $\mathbf{Q}$ at time t, contradiction.

Inductive Step: Now, let $m > 0$ and assume that the m most probable minimal diagnoses w.r.t. DPI are already elements of $\mathbf{D}_{calc}$. Suppose further that some minimal diagnosis $\mathcal{D}$ w.r.t. DPI is added to $\mathbf{D}_{calc}$ where $\mathcal{D}$ is not the $(m+1)$-th most probable minimal diagnosis w.r.t. DPI. This implies that $\mathcal{D}$ is processed and that $\mathcal{D}$ has the highest probability as per $p_{nodes}()$ among all nodes that are elements of $\mathbf{Q}$

at time t.

Let us denote by $\mathcal{D}_{m+1}$ the $(m + 1)$-th most probable minimal diagnosis w.r.t. DPI. That is, $p_{nodes}(\mathcal{D}_{m+1}) > p_{nodes}(\mathcal{D})$ holds since the m most probable minimal diagnoses w.r.t. DPI are already elements of $\mathbf{Q}$.

Then, by Proposition 12.8, at any point in time during the execution of DYNAMICHS with the current DPI DPI, there must be some node nd_{m+1} such that $\mathsf{nd}_{m+1} \subseteq \mathcal{D}_{m+1}$ and nd_{m+1} is an element of one of the collections (a) $\mathbf{D}_{calc}$, (b) $\mathbf{D}_{\checkmark}$, (c) $\mathbf{D}_{\times}$, (d) $\mathbf{D}_{\supset}$ or (e) $\mathbf{Q}$.

Case (a) can be ruled out due to proposition (1) which affirms that only minimal diagnoses w.r.t. DPI can be elements of $\mathbf{D}_{calc}$. As $\mathcal{D}_{m+1}$ is not an element of $\mathbf{D}_{calc}$ per assumption, a node $\mathsf{nd}_{m+1} = \mathcal{D}_{m+1}$ cannot be an element of $\mathbf{D}_{calc}$. Furthermore, by the fact that $\mathcal{D}_{m+1}$ is a minimal diagnosis w.r.t. DPI, any node $\mathsf{nd}_{m+1} \subset \mathcal{D}_{m+1}$ cannot be a (minimal) diagnosis w.r.t. DPI and thus cannot be an element of $\mathbf{D}_{calc}$. Cases (b)-(d) can be treated analogously as above in the proof of proposition (1). Hence, case (e) must hold.

That is, $\mathsf{nd}_{m+1} \in \mathbf{Q}$ at time t and nd_{m+1} is equal to or a subset of $\mathcal{D}_{m+1}$. As $p_{nodes}(\mathsf{nd}_{m+1}) \geq p_{nodes}(\mathcal{D}_{m+1}) > p_{nodes}(\mathcal{D})$ holds by Lemma 4.14, we can infer that $\mathcal{D}$ has not the highest probability as per $p_{nodes}()$ among all nodes that are elements of $\mathbf{Q}$ at time t, contradiction. $\qquad\square$

12.4.10 Correctness of DYNAMICHS

Now, we are able to prove that DYNAMICHS terminates and yields an output complying with the assertions given in Algorithm 8:

Corollary 12.8. *Any call to* DYNAMICHS *(given the inputs described in Algorithm 8) within Algorithm 5 terminates and yields an output* $\langle \mathbf{D}_{calc}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times}, \mathbf{D}_{\supset}, \mathbf{Q}_{dup} \rangle$ *where*

(1) $\mathbf{D}_{calc}$ *is the current set of leading diagnoses such that*

> *(a)* $\mathbf{D}_{calc} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ *is the set of most probable minimal diagnoses w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ *such that*
>
> > *(i)* $n_{\min} \leq |\mathbf{D}_{calc}| \leq n_{\max}$ *and*
> > *(ii)* $\mathbf{D}_{calc} \setminus \mathbf{D}_{\checkmark} \neq \emptyset$,
>
> *if such a set* $\mathbf{D}_{calc}$ *exists; or*
>
> *(b)* $\mathbf{D}_{calc}$ *is equal to the set of all minimal diagnoses* $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$, *otherwise;*
>
> *where "most-probable" refers to the probability measure* $p_{nodes}()$ *given by Definition 4.9 and obtained from the function* $p()$ *given as an input argument to* DYNAMICHS.

(2) $\mathbf{Q}$ *is the current queue of open (non-labeled) nodes of the produced hitting set tree,*

(3) $\mathbf{C}_{calc}$ *is a set of conflict sets w.r.t. the current DPI* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$,

(4) $\mathbf{D}_{\times} = \emptyset$,

(5) $\mathbf{D}_{\supset}$ *is the set of all processed nodes so far throughout the execution of Algorithm 5 that are non-minimal diagnoses w.r.t. the current DPI* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ *and*

(6) $\mathbf{Q}_{dup}$ *includes a node set-equal to* X *for a set* $X \subseteq \mathcal{K}$ *iff*

> • $\mathsf{nd} = X$ *is a generated node that is de-facto non-redundant w.r.t. the current DPI* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$, *such that, at generation time of* nd, *there was a node set-equal to* X *in* $\mathbf{Q}$ *or*
>
> • *there is a de-facto non-redundant node* $\mathsf{nd}' = X$ *w.r.t. the current DPI* $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ *which is a combined equal node of some generated node* nd'' *that has been added to* $\mathbf{Q}_{dup}$.

Proof. First, we prove that any call to DYNAMICHS within Algorithm 5 terminates. To this end, assume that a call to DYNAMICHS executes infinitely. That is, $\mathbf{Q} = [\,]$ must not be satisfied at any time during the execution of DYNAMICHS due to the stop criterion of DYNAMICHS in line 24.

However, the overall number of nodes that might be elements of $\mathbf{Q}$ during the processing of the repeat-loop of any call to DYNAMICHS is finite. This is satisfied since each node nd in DYNAMICHS is a list corresponding to a subset of $\mathcal{K}$ and each element of the list nd.cs is a subset of $\mathcal{K}$ as well. For, a node can never correspond to a proper superset of $\mathcal{K}$ by Proposition 4.9 which says that $QX(\langle \mathcal{K} \setminus \mathcal{D}, \mathcal{B}, P \cup P', N \cup N' \rangle_R)$ returns 'no conflict' in case $\mathcal{K} \setminus \mathcal{D}$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ which is equivalent to $\mathcal{D}$ being a diagnosis w.r.t. $\langle \mathcal{K} \setminus \mathcal{D}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ by Corollary 3.3. Now, the DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ is admissible which follows from the admissibility of the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ and Corollary 7.3. That $\mathcal{D} := \mathcal{K}$ must be a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ is a direct consequence of the admissibility of $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ and Definition 3.6. Therefore DLABEL must return *valid* for each node the latest when the node becomes set-equal to $\mathcal{K}$. A node that was assigned the label *valid* and added to $\mathbf{D}_{calc}$ can never be processed again during this execution of DYNAMICHS wherefore no successors of such a node can be added to $\mathbf{Q}$. The same holds for some node that is labeled by *nonmin* and added to $\mathbf{D}_{\supset}$.

Thence, the assumption that $\mathbf{Q} \neq [\,]$ forever implies that there is (at least) one node node that is never removed from $\mathbf{Q}$.

By Lemma 12.12, each node that is a subset of or set-equal to a once processed node nd must have been generated before nd is processed. That is, after a node is processed, it is guaranteed that no proper subsets of it can ever be processed and no subsets of it can ever be added to $\mathbf{Q}$. After a node nd is processed and is not labeled by *valid* or *nonmin*, nd is not an element of $\mathbf{Q}$ anymore (cf. line 7) and $\mathbf{Q}$ comprises a set of successor nodes of nd where each such node corresponds to a proper superset of nd (cf. line 23). Consequently, a node in $\mathbf{Q}$ that is processed can either be deleted whereupon no successor thereof is added to $\mathbf{Q}$ (in case of pruning or labeling a node by *valid* or *nonmin*) or be deleted whereupon proper supersets of it are added to $\mathbf{Q}$ (in case of labeling a node by a conflict set).

A (combined) replacement of a node involves the substitution of this node by another node set-equal to it. However, there can be only finitely many possibilities to construct a replacement or combined replacement node of some node since $Comb(\mathbf{Q}_{dup}) \supseteq \mathbf{Q}_{dup}$ also includes only nodes, i.e. finitely many elements. Therefore, each node in $\mathbf{Q}$ can be replaced only finitely many times.

Since in each iteration of the repeat-loop in DYNAMICHS one node is processed, the cardinality of the nodes that are elements of $\mathbf{Q}$ is strictly monotonically increasing.

As node is supposed to be never processed, we have that in each iteration of the repeat-loop, one of the other nodes in $\mathbf{Q}$ must by processed. By the given argumentation, we know that after finitely many iterations, $\mathbf{Q} = [\text{node}]$ must be given (since all other nodes must be already pruned or labeled). Hence, node will be processed in the next iteration as GETFIRST in line 6 must catch node, contradiction.

Proposition (1): This proposition is a direct consequence of Proposition 12.9-(2) and the stop criterion of DYNAMICHS in line 24.

Proposition (2) is clear. Proposition (3) follows from Lemma 12.2 which asserts that each element of $\mathbf{C}_{calc}$ is a minimal conflict set w.r.t. some DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P'', N \cup N'' \rangle_R$ where $P'' \subseteq P'$ and $N'' \subseteq N'$. By Proposition 12.1, we obtain that each element of $\mathbf{C}_{calc}$ is a conflict set w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$.

Proposition (4): This proposition is *true* since UPDATETREE is called at the beginning of each execution of DYNAMICHS and all elements in $\mathbf{D}_{\times}$ that have not been deleted from $\mathbf{D}_{\times}$ before are deleted in lines 67-69. After UPDATETREE has finished processing, there is no other place in DYNAMICHS where nodes can be added to $\mathbf{D}_{\times}$. Hence, $\mathbf{D}_{\times} = \emptyset$ must hold when DYNAMICHS terminates.

Proposition (5): The elements of $\mathbf{D}_{\supset}$ after UPDATETREE at the beginning of the execution of DYNAMICHS has returned must be non-minimal diagnoses w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ by lines 70-78 and the fact that $\mathbf{D}_{\checkmark}$ comprises only diagnoses w.r.t. the current DPI. The latter holds by

lines 19 and 21 of Algorithm 5 where only diagnoses w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ are added to $\mathbf{D}_{\checkmark}$. That only non-minimal diagnoses w.r.t. the current DPI can be added to $\mathbf{D}_{\supset}$ during the execution of the repeat-loop is a simple implication of Lemma 12.1-(4).

Proposition (6) is a consequence of lines 20-21, the definition of de-facto non-redundancy (Definition 12.7) and Lemma 12.8. $\qquad \square$

Algorithm 8 Iterative Construction of a Dynamic Hitting Set Tree

---

Input: a tuple $\langle \langle \mathcal{K}, \mathcal{B}, P, N \rangle_R, \mathbf{Q}, \mathbf{Q}_{dup}, t, n_{\min}, n_{\max}, \mathbf{C}_{calc}, \mathbf{D}_{\checkmark}, \mathbf{D}_{\times}, p(), P', N', \mathbf{D}_{\supset} \rangle$ consisting of

- the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ given as input to Algorithm 5,

- the overall sets of positively (P') and negatively (N') answered queries added as test cases to $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ so far,

- a queue $\mathbf{Q}$ of open (non-labeled) nodes,

- some computation timeout t,

- a desired minimal ($n_{\min} \geq 2$) and maximal ($n_{\max}$) number of minimal diagnoses to be returned,

- a set $\mathbf{C}_{calc}$ of conflict sets w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$,

- a set $\mathbf{D}_{\checkmark}$ of minimal diagnoses w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$,

- a set $\mathbf{D}_{\times}$ of minimal diagnoses w.r.t. the last-but-one DPI that are invalidated by the most recently added test case,

- a function $p : \mathcal{K} \rightarrow (0, 0.5)$,

- a set $\mathbf{D}_{\supset}$ of non-minimal diagnoses w.r.t. the last-but-one DPI and

- a set $\mathbf{Q}_{dup}$ of stored (duplicate) nodes nd that can be used when it comes to constructing a replacement node of a pruned node nd$' \supseteq$ nd after tree pruning.

Output: a tuple $\langle \mathbf{D}_{calc}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times}, \mathbf{D}_{\supset} \rangle$ where

- $\mathbf{D}_{calc}$ is the current set of leading diagnoses such that

    (a) $\mathbf{D}_{calc} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ is the set of most probable minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ such that

        (i) $n_{\min} \leq |\mathbf{D}_{calc}| \leq n_{\max}$ and

        (ii) $\mathbf{D}_{calc} \setminus \mathbf{D}_{\checkmark} \neq \emptyset$,

        if such a set $\mathbf{D}_{calc}$ exists, or

    (b) $\mathbf{D}_{calc}$ is equal to the set of all minimal diagnoses $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$, otherwise,

    where "most-probable" refers to the probability measure $p_{nodes}()$ (cf. Definition 4.9) obtained from the given function $p()$;

- $\mathbf{Q}$ is the current queue of open (non-labeled) nodes of the hitting set tree,

- $\mathbf{C}_{calc}$ is a set of conflict sets w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$,

- $\mathbf{D}_{\times} = \emptyset$,

- $\mathbf{D}_{\supset}$ is the set of all processed nodes so far throughout the execution of Algorithm 5 that are non-minimal diagnoses w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ and

- $\mathbf{Q}_{dup}$ includes a node set-equal to X for a set $X \subseteq \mathcal{K}$ iff

    - nd $= X$ is a generated node that is de-facto non-redundant w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$, such that, at generation time of nd, there was a node set-equal to X in $\mathbf{Q}$ or

    - there is a de-facto non-redundant node nd$' = X$ w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ which is a combined equal node of some generated node nd$''$ that has been added to $\mathbf{Q}_{dup}$.

1: **procedure** DYNAMICHS($\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R, \mathbf{Q}, \mathbf{Q}_{dup}, t, n_{\min}, n_{\max}, \mathbf{C}_{calc}, \mathbf{D}_{\checkmark}, \mathbf{D}_{\times}, p(), P', N', \mathbf{D}_{\supset}$)
2:     $t_{start} \leftarrow$ GETTIME()
3:     $\mathbf{D}_{calc} \leftarrow \emptyset$
4:     $\langle \mathbf{Q}, \mathbf{D}_{\times}, \mathbf{D}_{\supset}, \mathbf{C}_{calc}, \mathbf{Q}_{dup} \rangle \leftarrow$ UPDATETREE($\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R, \mathbf{D}_{\times}, \mathbf{Q}, \mathbf{Q}_{dup}, \mathbf{D}_{\supset}, \mathbf{D}_{\checkmark}, \mathbf{C}_{calc}, p(), P', N'$)
5:     **repeat**   ▷ UPDATETREE (see Algorithm 9)
6:         node $\leftarrow$ GETFIRST($\mathbf{Q}$)   ▷ node is processed
7:         $\mathbf{Q} \leftarrow$ DELETEFIRST($\mathbf{Q}$)
8:         **if** node $\in \mathbf{D}_{\checkmark}$ **then**   ▷ $\mathbf{D}_{\checkmark}$ includes only minimal diagnoses w.r.t. current DPI
9:             $L \leftarrow valid$
10:        **else**
11:            $\langle L, \mathbf{C}_{calc}, \mathbf{Q}_{dup} \rangle \leftarrow$ DLABEL($\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, node, $\mathbf{C}_{calc}, \mathbf{D}_{calc}, \mathbf{Q}, \mathbf{Q}_{dup}, p(), P', N'$)
12:        **if** $L = valid$ **then**   ▷ DLABEL (see Algorithm 9)
13:            $\mathbf{D}_{calc} \leftarrow \mathbf{D}_{calc} \cup \{$node$\}$ ▷ node is a minimal diagnosis w.r.t. current DPI
14:        **else if** $L = nonmin$ **then**
15:            $\mathbf{D}_{\supset} \leftarrow \mathbf{D}_{\supset} \cup \{$node$\}$ ▷ node is a non-minimal diagnosis w.r.t. current DPI
16:        **else**
17:            **for** $e \in L$ **do**   ▷ L is a minimal conflict set w.r.t. current DPI
18:                node$_e \leftarrow$ ADD(node, e)   ▷ node$_e$ is generated
19:                node$_e$.cs $\leftarrow$ ADD(node.cs, L)
20:                **if** node$_e \in \mathbf{Q}$ **then**   ▷ node$_e$ is a (set-equal) duplicate of a node in $\mathbf{Q}$
21:                    $\mathbf{Q}_{dup} \leftarrow$ INSERTSORTED(node$_e, \mathbf{Q}_{dup}, cardinality, ascending$)
22:                **else**
23:                    $\mathbf{Q} \leftarrow$ INSERTSORTED(node$_e, \mathbf{Q}, p_{nodes}(), descending$)
24:     **until** $\mathbf{Q} = [] \vee [\mathbf{D}_{calc} \setminus \mathbf{D}_{\checkmark} \neq \emptyset \wedge |\mathbf{D}_{calc}| \geq n_{\min} \wedge (|\mathbf{D}_{calc}| = n_{\max} \vee$ GETTIME() $- t_{start} > t)]$
25:     **return** $\langle \mathbf{D}_{calc}, \mathbf{Q}, \mathbf{C}_{calc}, \mathbf{D}_{\times}, \mathbf{D}_{\supset}, \mathbf{Q}_{dup} \rangle$

Algorithm 9 Iterative Construction of a Dynamic Hitting Set Tree (continued)

26: **procedure** DLABEL($\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$, node, $\mathbf{C}_{calc}$, $\mathbf{D}_{calc}$, $\mathbf{Q}$, $\mathbf{Q}_{dup}$, $p()$, P', N')   ▷ DLABEL (see page 196)
27:     **for** nd $\in \mathbf{D}_{calc}$ **do**
28:       **if** node $\supset$ nd **then**   ▷ node is a non-minimal diagnosis
29:         **return** $\langle nonmin, \mathbf{C}_{calc}, \mathbf{Q}_{dup} \rangle$
30:     **for** $\mathcal{C} \in \mathbf{C}_{calc}$ **do**   ▷ $\mathbf{C}_{calc}$ includes only conflict sets w.r.t. current DPI
31:       **if** $\mathcal{C} \cap$ node $= \emptyset$ **then**   ▷ reuse (a subset of) $\mathcal{C}$ to label node
32:         $X \leftarrow$ QX($\langle \mathcal{C}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$)   ▷ Algorithm 1 (page 48) to test if $\mathcal{C}$ is minimal w.r.t. current DPI
33:         **if** $X = \mathcal{C}$ **then**
34:           **return** $\langle \mathcal{C}, \mathbf{C}_{calc}, \mathbf{Q}_{dup} \rangle$
35:         **else**   ▷ $X \subset \mathcal{C}$
36:           $\mathbf{Q}_{dup} \leftarrow$ PRUNEQDUP($X, \mathbf{Q}_{dup}$)   ▷ PRUNEQDUP (see Algorithm 10)
37:           $\mathbf{Q} \leftarrow$ PRUNE($X, \mathbf{Q}, \mathbf{Q}_{dup}, p_{nodes}()$)   ▷ PRUNE (see Algorithm 10)
38:           $\mathbf{D}_{\supset} \leftarrow$ PRUNE($X, \mathbf{D}_{\supset}, \mathbf{Q}_{dup}, \emptyset$)
39:           $\mathbf{C}_{calc} \leftarrow$ ADDSETDELSUPSETS($X, \mathbf{C}_{calc}$)   ▷ add X to $\mathbf{C}_{calc}$ and delete all its supersets from $\mathbf{C}_{calc}$
40:           **return** $\langle X, \mathbf{C}_{calc}, \mathbf{Q}_{dup} \rangle$
41:     $L \leftarrow$ QX($\langle \mathcal{K} \setminus$ node, $\mathcal{B}, P \cup P', N \cup N' \rangle_R$)   ▷ Algorithm 1 (page 48) to test if node is a diagnosis
42:     **if** $L =$ 'no conflict' **then**   ▷ node is a diagnosis
43:       **return** $\langle valid, \mathbf{C}_{calc}, \mathbf{Q}_{dup} \rangle$
44:     **else**   ▷ L is a *new* minimal conflict set ($\notin \mathbf{C}_{calc}$)
45:       $\mathbf{C}_{calc} \leftarrow \mathbf{C}_{calc} \cup \{L\}$
46:       **return** $\langle L, \mathbf{C}_{calc}, \mathbf{Q}_{dup} \rangle$

47: **procedure** UPDATETREE($\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R, \mathbf{D}_{\times}, \mathbf{Q}, \mathbf{Q}_{dup}, \mathbf{D}_{\supset}, \mathbf{D}_{\checkmark}, \mathbf{C}_{calc}, p(), P', N'$)
48:     **for** nd $\in \mathbf{D}_{\times}$ **do**
49:       $quickRC, completeRC \leftarrow false$
50:       $X \leftarrow$ QX($\langle U_{\mathsf{nd.cs}} \setminus$ nd, $\mathcal{B}, P \cup P', N \cup N' \rangle_R$)   ▷ QRC begin
51:       **for** $\mathcal{C} \in$ nd.cs **do**
52:         **if** $X \subset \mathcal{C}$ **then**   ▷ QRC (see page 205)
53:           $quickRC \leftarrow true$
54:           **break**   ▷ QRC end
55:       **if** $quickRC = false$ **then**   ▷ CRC begin
56:         **for** $i \leftarrow 1, \ldots, |nd|$ **do**
57:           $X \leftarrow$ QX($\langle$nd.cs$[i] \setminus \{$nd$[i]\}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$)   ▷ CRC (see page 206)
58:           **if** $X \neq$ 'no conflict' **then**
59:             $completeRC \leftarrow true$
60:             **break**   ▷ CRC end
61:       **if** $quickRC = true \lor completeRC = true$ **then**   ▷ condition *true* iff nd redundant w.r.t. current DPI
62:         $\mathbf{Q}_{dup} \leftarrow$ PRUNEQDUP($X, \mathbf{Q}_{dup}$)   ▷ PRUNEQDUP (see Algorithm 10)
63:         $\mathbf{Q} \leftarrow$ PRUNE($X, \mathbf{Q}, \mathbf{Q}_{dup}, p_{nodes}()$)   ▷ PRUNE (see Algorithm 10)
64:         $\mathbf{D}_{\times} \leftarrow$ PRUNE($X, \mathbf{D}_{\times}, \mathbf{Q}_{dup}, \emptyset$)
65:         $\mathbf{D}_{\supset} \leftarrow$ PRUNE($X, \mathbf{D}_{\supset}, \mathbf{Q}_{dup}, \emptyset$)
66:         $\mathbf{C}_{calc} \leftarrow$ ADDSETDELSUPSETS($X, \mathbf{C}_{calc}$)   ▷ add X to $\mathbf{C}_{calc}$ and delete all its supersets from $\mathbf{C}_{calc}$
67:     **for** nd $\in \mathbf{D}_{\times}$ **do**   ▷ add all (non-pruned) nodes in $\mathbf{D}_{\times}$ to $\mathbf{Q}$
68:       $\mathbf{Q} \leftarrow$ INSERTSORTED(nd, $\mathbf{Q}, p_{nodes}(), descending$)
69:       $\mathbf{D}_{\times} \leftarrow \mathbf{D}_{\times} \setminus \{nd\}$
70:     **for** nd $\in \mathbf{D}_{\supset}$ **do**   ▷ update $\mathbf{D}_{\supset}$: add all nodes to $\mathbf{Q}$ which are not proper supersets of a diagnosis in $\mathbf{D}_{\checkmark}$
71:       $nonmin \leftarrow false$
72:       **for** nd$' \in \mathbf{D}_{\checkmark}$ **do**
73:         **if** nd $\supset$ nd$'$ **then**
74:           $nonmin \leftarrow true$
75:           **break**
76:       **if** $nonmin = false$ **then**
77:         $\mathbf{Q} \leftarrow$ INSERTSORTED(nd, $\mathbf{Q}, p_{nodes}(), descending$)
78:         $\mathbf{D}_{\supset} \leftarrow \mathbf{D}_{\supset} \setminus \{nd\}$
79:     **for** $\mathcal{D} \in \mathbf{D}_{\checkmark}$ **do**   ▷ reinsert known minimal diagnoses to $\mathbf{Q}$ to find diagnoses in order of descending $p_{nodes}()$
80:       $\mathbf{Q} \leftarrow$ INSERTSORTED($\mathcal{D}, \mathbf{Q}, p_{nodes}(), descending$)
81:     **return** $\langle \mathbf{Q}, \mathbf{D}_{\times}, \mathbf{D}_{\supset}, \mathbf{C}_{calc}, \mathbf{Q}_{dup} \rangle$

Algorithm 10 Iterative Construction of a Dynamic Hitting Set Tree (continued)

82: **procedure** PRUNE($X, S, Dup, sort\_measure$)   ▷ PRUNE (see page 212)
83:   **if** S is a list **then**
84:     $S' \leftarrow [\,]$
85:   **else**
86:     $S' \leftarrow \emptyset$
87:   **for** nd $\in S$ **do**
88:     $k \leftarrow 0$
89:     **for** $i = 1$ to $|$nd.cs$|$ **do**
90:       **if** nd.cs$[i] \supset X$ **then**   ▷ check first redundancy criterion (Definition 12.4 on page 202)
91:         **if** nd$[i] \in$ nd.cs$[i] \setminus X$ **then**   ▷ check second redundancy criterion (Definition 12.4 on page 202)
92:           $k \leftarrow i$
93:         **else**
94:           nd.cs$[i] \leftarrow X$ ▷ replace each superset of X in nd.cs by X
95:     **if** $k > 0$ **then**   ▷ nd is redundant
96:       **for** node $\leftarrow Dup[1], \dots, Dup[|Dup|]$ **do**
97:         **if** $|$node$| \geq k \wedge$ nd$[1..|$node$|] =$ node **then**
98:           nd$_{new} \leftarrow$ ADD(node, nd$[|$node$| + 1..|$nd$|]$)   ▷ construct replacement node nd$_{new}$ of nd
99:           nd$_{new}.cs \leftarrow$ ADD(node.cs, nd.cs$[|$node$| + 1..|$nd$|]$)
100:           $S' \leftarrow$ INSERTSORTED(nd$_{new}, S', sort\_measure, descending$)
101:           **break**
102:       **else**   ▷ X is not a witness of redundancy of nd
103:         $S' \leftarrow$ INSERTSORTED(nd, $S', sort\_measure, descending$)
104:   **return** S'

105: **procedure** PRUNEQDUP(X, Dup)   ▷ PRUNEQDUP (see page 209)
106:   $Dup_{new} \leftarrow [\,]$
107:   **for** $i \leftarrow 1$ to $|Dup|$ **do**
108:     ndi $\leftarrow Dup[i]$
109:     $k \leftarrow 0$
110:     **for** $m \leftarrow 1$ to $|$ndi.cs$|$ **do**
111:       **if** ndi.cs$[m] \supset X$ **then**   ▷ check first redundancy criterion (Definition 12.4 on page 202)
112:         **if** ndi$[m] \in$ ndi.cs$[m] \setminus X$ **then**   ▷ check second redundancy criterion (Definition 12.4 on page 202)
113:           $k \leftarrow m$
114:         **else**
115:           ndi.cs$[m] \leftarrow X$ ▷ replace each superset of X in ndi.cs by X
116:     **if** $k > 0$ **then**   ▷ ndi is redundant
117:       **for** ndj $\in Dup_{new}$ **do**
118:         **if** $|$ndj$| \geq k \wedge$ ndi$[1..|$ndj$|] =$ ndj **then**
119:           ndi$_{new} \leftarrow$ ADD(ndj, ndi$[|$ndj$| + 1..|$ndi$|]$)   ▷ construct combined replacement node ndi$_{new}$ of ndi
120:           ndi$_{new}.cs \leftarrow$ ADD(ndj.cs, ndi.cs$[|$ndj$| + 1..|$ndi$|]$)
121:           $Dup_{new} \leftarrow$ INSERTSORTED(ndi$_{new}, Dup_{new}, cardinality, ascending$)
122:           **break**
123:       **else**   ▷ X is not a witness of redundancy of ndi
124:         $Dup_{new} \leftarrow$ INSERTSORTED(ndi, $Dup_{new}, cardinality, ascending$)
125:   **return** Dup_{new}

Chapter 13

Discussion of Iterative Diagnosis Computation

In this chapter we want to summarize properties of and differences between STATICHS and DYNAMICHS that we already pointed out in previous sections and, additionally, we want to shed light on some further interesting aspects of these iterative diagnosis computation methods in the scope of interactive KB debugging (Algorithm 5). Table 13.1 provides an overview of what we did discuss or will discuss below.

First Segment of Table 13.1 – Addressed Problem and Properties w.r.t. Solutions. The first row of the table has been proven by Proposition 9.1 on page 124. Results given by the second up to the fourth row of the table are substantiated by Proposition 11.1 (STATICHS) and Corollary 12.8 (DYNAMICHS). We have discussed in Section 11.1 that Algorithm 5 with $mode = static$ can artificially fix the search space for possible solutions initially. This is an inherent property of the Interactive Static KB Debugging Problem which the algorithm aims to solve in static mode. For, a minimal diagnosis w.r.t. the input DPI which satisfies all answered queries added as test cases throughout the debugging session must be detected (see left column of category "diagnoses" in Table 13.1). Hence, the solution space is given by $|\mathbf{mD}_{inputDPI}|$. "Initially fixed search space" in this case means that, given the fault tolerance $\sigma = 0$, Algorithm 5 in static mode must compute all minimal diagnoses w.r.t. the input DPI, i.e. the entire set $\mathbf{mD}_{inputDPI}$. In case of dynamic mode, on the other hand, the solution space (i.e. minimal diagnoses w.r.t. the current DPI, see right column of Table 13.1 in category "diagnoses") that needs to be explored by Algorithm 5 for a given value of zero for σ is not known in advance. It rather depends on which test cases are specified or, respectively, which queries the user is asked. In case of the usage of mainly "positive-impact queries", the search space might have significantly smaller cardinality than $\mathbf{mD}_{inputDPI}$ whereas it might grow significantly beyond the cardinality of $\mathbf{mD}_{inputDPI}$ in a scenario where many unfavorable "negative-impact queries" are generated (cf. Section 12.1). The maximum theoretically possible cardinality of the search space for DYNAMICHS is given by $|\mathbf{aD}_{inputDPI}|$ due to Corollary 12.4.

Second Segment of Table 13.1 – Impact of New Test Cases and Computation Focus. The properties given in the category "computes" in Table 13.1 are confirmed by Proposition 11.1 (STATICHS) and Corollary 12.8 (DYNAMICHS). Hence, other than DYNAMICHS which analyzes the current DPI in terms of minimal conflict sets and diagnoses in each iteration, STATICHS must only consider minimal conflict sets w.r.t. the *input DPI* (see categories "diagnoses" and "conflict sets" in Table 13.1). This is sufficient for the exploration of all minimal diagnoses w.r.t. the input DPI by Proposition 4.6. In this vein, new test cases in static KB debugging are not taken into account in the computation of minimal

conflict sets. Instead, new test cases are just exploited to invalidate *already computed* minimal diagnoses w.r.t. the input DPI. Thus, test cases specified *during* static KB debugging are treated somewhat inferior to test cases already present in the input DPI. Because, the newly gained information given by these test cases is not utilized to reveal new faults in the KB or to lay the focus on just the *now* relevant parts of existing faults, but only for the purpose of constraining the search space for minimal diagnoses w.r.t. the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. We might thus call test cases added during the execution of Algorithm 5 with $mode = static$ pure *differentiation test cases* (see category "purpose of test cases" in Table 13.1).

Of course, seen from the point of view of a current DPI, i.e. the input DPI extended by differentiation test cases, STATICHS does not guarantee completeness w.r.t. this current DPI, but only w.r.t. the initial one. This however does not mean that, after the (exact) solution $\mathcal{K}^* := (\mathcal{K} \setminus \mathcal{D}) \cup U_P$ of the Interactive Static KB Debugging problem has been localized by means of STATICHS, the differentiation test cases (P' and N') cannot be simply added to the DPI. In this case, $\mathcal{K}^*$ is *still* a maximal solution KB w.r.t. the extended input DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. In other words, there is no conflict set (and thus no diagnosis) w.r.t. $\langle \mathcal{K} \setminus \mathcal{D}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ and $\mathcal{K} \setminus \mathcal{D}$ is valid w.r.t. $\langle \cdot, \mathcal{B}, P \cup P', N \cup N' \rangle_R$. However, in spite of using the (exact) solution KB of the Interactive Static KB Debugging problem, it is not ensured that this solution is the optimal one w.r.t. the *extended DPI*, i.e. of the Interactive Dynamic KB Debugging problem. This is because user interaction is just exploited to the extent that the best solution w.r.t. the input DPI is crystallized out. It is not used to have the solution verified by the user in the light of the extended DPI.

On the other hand, test cases assigned throughout dynamic KB debugging by means of Algorithm 5 with $mode = dynamic$ are treated equally as test cases already given in the input DPI. They are used to prune the search space and to pinpoint new faults that arise from added test cases resulting from answered queries. The dynamic algorithm assists the user in filtering out a solution and verifying in a thorough manner that this solution is the desired one w.r.t. the extended DPI, among *all* existing solutions w.r.t. the extended DPI. Due to these aspects we might regard Algorithm 5 with mode $mode = dynamic$ as the *standard method for Interactive KB Debugging*.

In Sections 11.1, 12.1, 12.4.3 and 12.4.4 we have thoroughly investigated the impact of new test cases (answered queries) added to the DPI on the set of minimal (all) diagnoses and the set of minimal conflict sets considered by the respective method STATICHS or DYNAMICHS. For the former, we have shown that (for arbitrary iteration i of Algorithm 5) $\mathbf{mD}_i \supset \mathbf{mD}_{i+1}$ and $\mathbf{aD}_i \supset \mathbf{aD}_{i+1}$ where $\mathbf{mD}_i$ and $\mathbf{aD}_i$ denote the set of all minimal diagnoses and the set of all diagnoses, respectively, that are relevant (for the DPI considered) during iteration i. That is, the set of minimal as well as the set of all diagnoses (w.r.t. the input DPI) is reduced to a proper subset after a new test case has been added. For the latter, (for arbitrary iteration i of Algorithm 5) we have argued that generally $\mathbf{mD}_i \not\supset \mathbf{mD}_{i+1}$, but still $\mathbf{aD}_i \supset \mathbf{aD}_{i+1}$, where $\mathbf{mD}_i$ and $\mathbf{aD}_i$ are defined as above. That is, not only might some minimal diagnoses (w.r.t. the last-but-one DPI) be invalidated, but also some new ones (w.r.t. the current DPI) might originate from the incorporation of the information given by a query answer.

Concerning minimal conflict sets, the set of all (or: relevant) minimal conflict sets does not change throughout a debugging session by means of STATICHS, i.e. $\mathbf{mC}_i = \mathbf{mC}_{i+1}$ (for arbitrary iteration i of Algorithm 5) where $\mathbf{mC}_i$ is the set of minimal conflict sets relevant (for the DPI considered) during iteration i. This holds since the minimal conflict sets w.r.t. the input DPI are artificially fixed (see above). On the contrary, the assignment of a new test case using DYNAMICHS involves the reduction of some minimal conflict sets (w.r.t. the last-but-one DPI) to smaller subset conflict sets (w.r.t. the current DPI) and/or the introduction of some "completely new" minimal conflict sets (which are in no subset-relation with existing ones, cf. Section 12.1). These results are summarized by the categories "set of all X upon addition of a test case" in Table 13.1.

Third Segment of Table 13.1 – Hitting Set Tree Construction, Pruning and Complexity.   Regarding the constructed hitting set tree, we have explained that STATICHS builds a wpHS-tree (see Definition 4.10

on page 74 and the argumentation in Section 11.4) just as the HS method which is employed for diagnosis computation in the presented non-interactive KB debugging scenario (Algorithm 3). The main differences between Algorithm 5 in static mode and Algorithm 3 are, first, that the former constructs the wpHS-tree step-by-step in multiple phases. Between each two phases a query is generated and presented to the user. The latter, by contrast, finishes the tree construction (to the extent as prescribed by the given parameters $n_{\min}$, $n_{\max}$ and t, see Section 4.7) before a single most probable automatically selected solution or a set of solutions is displayed to the user. Second, the tree constructed by the interactive static algorithm exhibits a different labeling of leaf nodes than the one built up be the non-interactive algorithm. In the former, some leaf nodes might be labeled by $\times$ indicating that the path to this node is a minimal diagnosis w.r.t. the input DPI, but one which is not in accordance with all answered queries. Notice that such invalidated diagnoses cannot be simply deleted in favor of memory savings, but must be stored in order for the non-minimality criterion (lines 21-23) to function properly which is necessary to preserve the property of STATICHS to compute only *minimal* diagnoses (cf. Lemma 11.7). In the non-interactive wpHS-tree, on the other hand, all minimal diagnoses w.r.t. the input DPI are labeled by $\checkmark$.

What the interactive static and the non-interactive tree have in common is the usage of only minimal conflict sets w.r.t. the input DPI as labels of internal (i.e. non-leaf) nodes and the adherence to the "standard" pruning rules [Rei87] as per Definition 4.8 on page 59, i.e. the immediate deletion of non-minimal and duplicate tree paths. Except for the standard pruning actions that take place during tree expansion, no separate pruning phases are performed by STATICHS. The reason for this is the fixation of the minimal conflict sets, i.e. the consideration of only minimal conflict sets w.r.t. the input DPI. Incorporation of new minimal conflict sets resulting from answered queries would generally negate completeness of STATICHS w.r.t. the exploration of all minimal diagnoses w.r.t. the input DPI. Integration of new conflict sets that are subsets of existing ones, however, is the key to more substantial pruning actions carried out by DYNAMICHS.

Due to the more or less equivalent construction of both the tree built up by STATICHS and the one constructed by the HS method in the non-interactive algorithm, it is straightforward to recognize that the worst case time and space complexity of both *tree* computations (without taking into the account other actions performed by the interactive algorithm like probability updates and query generations) are equal. By worst case complexity we refer to the complexity of the search for the (exact) solution of the Interactive Static KB Debugging Problem on the one hand and the complexity of enumerating all minimal diagnoses w.r.t. the input DPI on the other hand. In particular, the complexity of tree construction in static KB debugging is independent of given parameters such as the ones for leading diagnoses computation ($n_{\min}$, $n_{\max}$ and t) and of the test cases that are classified positively or negatively, respectively, during the debugging session.

To sum up, due to the artificial fixation of the solution set, there is no possibility of tree pruning in static KB debugging except for the standard pruning rules and hence no way to escape the generally immense worst case complexity for diagnosis search in case $\sigma = 0$.

The hitting set tree constructed by DYNAMICHS, on the other hand, might differ significantly from the wpHS-tree produced by the non-interactive algorithm. First, it uses minimal conflict sets w.r.t. the *current DPI* to label internal nodes in the tree during each expansion stage. Since minimal conflict sets can only "shrink" and not "grow" due to the integration of test cases into a DPI as stated by Proposition 12.1, the finding that by now a subset of a former minimal conflict set (w.r.t. some previous DPI) is already a minimal conflict set (w.r.t. the current DPI) gives rise to very powerful ways of tree pruning, as we detailed in Section 12.4.6 and illustrated by Example 12.2. In this vein, the evolution of the tree produced by DYNAMICHS can be characterized by alternating expansion and pruning stages. A pruning stage takes place after a test case has been added to the last-but-one DPI in order to modify the tree T_i used to search for minimal diagnoses w.r.t. the last-but-one DPI to obtain a tree T_{i+1} that enables the discovery of all minimal diagnoses w.r.t. the current DPI. Concretely, both pre-pruning as well as post-pruning is possible during a pruning phase. Pre-pruning refers to the deletion of tree paths ending in an open leaf node, i.e.

paths corresponding to partial diagnoses, and post-pruning refers to the deletion of tree paths ending in a closed node, i.e. paths corresponding to (minimal or non-minimal) diagnoses. Both pre- and post-pruning are not possible in STATICHS. The ability for significant tree pruning comes at the cost of not being able to exploit the standard pruning rules as STATICHS does. For, non-minimal diagnoses and duplicate tree paths must be stored to guarantee the proper working of tree pruning and in further consequence the completeness of minimal diagnoses search for each current DPI (see Section 12.4).

As we pointed out in Section 12.1, the test cases specified during the dynamic debugging session and the defined leading diagnoses computation parameters $n_{\min}$, $n_{\max}$ and t might have a material influence on the extent of possible tree pruning on the one hand and the extent of undesired tree growth on the other. Thence, worst case time and space complexity of the tree generation by means of DYNAMICHS cannot be initially (at least theoretically) quantified as in the case of STATICHS. Consequently, significant savings as well as a substantial overhead compared to STATICHS are possible. Careful "control" of certain properties of asked queries (added test cases) might help to keep considerable unwanted tree growth within bounds, as we touched upon in Section 12.1 and will elaborate on in future work.

Nevertheless, we want to mention a shortcoming of STATICHS compared to DYNAMICHS. Namely, for $\sigma = 0$, STATICHS must *enumerate all minimal diagnoses w.r.t. the input DPI* (otherwise no diagnosis can have a probability of 1, see the proof of Proposition 9.1 in Section 9.4) whereas DYNAMICHS might be able to obtain some extended DPI (by the addition of test cases) soon for which only one minimal diagnosis exists. This might require the computation of only a small fraction of the number of $|\mathbf{mD}_{inputDPI}|$ minimal diagnoses that STATICHS must determine and therefore might be substantially more time and space saving than figuring out all minimal diagnoses w.r.t. some DPI. This is quite well illustrated by Examples 11.2 and 12.2.

Fourth Segment of Table 13.1 – Query Generation and Bias. We explained in Remark 11.2 on page 153 that queries in STATICHS are computed w.r.t. the current DPI albeit only minimal diagnoses w.r.t. the input DPI (which are at the same time minimal diagnoses w.r.t. the current DPI, cf. bullet (a) on page 128) are considered and calculated by Algorithm 5 with $mode = static$. In the case of dynamic debugging it is clear that queries are computed w.r.t. the current DPI since only minimal diagnoses w.r.t. the current DPI are taken into account.

Another important property of an interactive KB debugging algorithm is whether it is biased or unbiased. Intuitively, we call an interactive KB debugging algorithm biased w.r.t. some current DPI DPI encountered during its execution iff there might be a minimal diagnosis $\mathcal{D}$ w.r.t. DPI such that $\mathcal{D}$ might be definitely invalidated independently of the answers a user gives. In other words, an interactive KB debugging algorithm is unbiased iff for each minimal diagnosis $\mathcal{D}$ w.r.t. DPI there is a set $QA_{\mathcal{D}}$ including query answer-pairs such that the addition of the positive queries in $QA_{\mathcal{D}}$ to the positive test cases of DPI and the addition of the negative queries in $QA_{\mathcal{D}}$ to the negative test cases of DPI yields an extended DPI DPI' such that $\mathcal{D}$ is the only minimal diagnosis w.r.t. DPI'. This means that unbiasedness implies that any solution w.r.t. any encountered current DPI during the debugging session might be found as the finally remaining (exact) solution diagnosis. So, all solutions are treated equitably by an unbiased algorithm and *only* the user may decide by their given answers which solutions are and which are not ruled out.

More formally, we define unbiasedness of an interactive KB debugging algorithm as follows:

Definition 13.1. *Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$ be the input DPI given to an algorithm Alg_X that solves the Interactive X Debugging Problem for $X \in \{static, dynamic\}$. Let $P' \supseteq \emptyset$ and $N' \supseteq \emptyset$ be the sets of test cases specified so far during the execution of Alg_X and let $\mathbf{D} \subseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ be the current set of leading diagnoses. Then, we call Alg_X biased w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ iff there is a diagnosis $\mathcal{D} \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ and a query $Q \in \mathbf{Q}_{\mathbf{D}, \langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ such that $\mathcal{D} \notin \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P' \cup \{Q\}, N \cup N' \rangle_R}$ and $\mathcal{D} \notin \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \cup \{Q\} \rangle_R}$.*

Further, we call Alg_X unbiased iff there cannot be any sets of test cases $P' \supseteq \emptyset$ and $N' \supseteq \emptyset$ during

any execution of Alg_X such that Alg_X is biased w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$.

Remark 13.1 It is important to notice the difference between completeness (which has already been established for Algorithm 5 using any of the methods STATICHS or DYNAMICHS, see Lemma 11.5 and Proposition 12.8) and unbiasedness of an algorithm. Completeness refers to the guarantee that the algorithm explores all minimal diagnoses w.r.t. any DPI DPI. However, it does not say anything about what might happen after a new test case Q is added to DPI. Although it does state that all minimal diagnoses w.r.t. the new DPI DPI' are explored, it leaves us unclear about what effect the addition of the query Q to the test cases might have had on the minimal diagnoses. So, there might be a minimal diagnosis w.r.t. DPI that would have been ruled out by both answers to Q thereby violating unbiasedness, but not completeness. To sum up, completeness gives us guarantees about what happens during the diagnosis computation phase whereas unbiasedness gives us guarantees about what happens during the transition from one DPI to a new DPI. $\square$

In the following, we show that Algorithm 5 in both static and dynamic mode is unbiased.

Proposition 13.1. *Assume the execution of Algorithm 5 with $mode \in \{static, dynamic\}$ given the input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$. Further, let $\mathbf{D} := \mathbf{D}_{calc}$ be the set of minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ returned by a call of DYNAMICHS in case of $mode = dynamic$ and $\mathbf{D} := \mathbf{D}_{calc} \cup \mathbf{D}_{\checkmark}$ be the set of minimal diagnoses w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ returned by a call of STATICHS in case of $mode = static$. Moreover, let $\mathcal{D} \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$.*

Then, no query Q w.r.t. $\mathbf{D}$ and $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ can be computed by Algorithm 5 such that $\mathcal{D} \notin \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P' \cup \{Q\}, N \cup N' \rangle_R}$ and $\mathcal{D} \notin \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \cup \{Q\} \rangle_R}$.

Proof. Let us consider the q-partition $\mathfrak{P}(Q) = \langle \mathbf{D}^+(Q), \mathbf{D}^-(Q), \mathbf{D}^0(Q) \rangle$ of the query Q that is computed by Algorithm 5 for the set of leading diagnoses $\mathbf{D}$. By Proposition 7.1, we have that $\mathbf{D}^+(Q) \cup \mathbf{D}^-(Q) \cup \mathbf{D}^0(Q) = \mathbf{D}$ and $\mathbf{D}^+(Q)$, $\mathbf{D}^-(Q)$ and $\mathbf{D}^0(Q)$ are pairwise disjoint sets, i.e. the sets $\mathbf{D}^+(Q)$, $\mathbf{D}^-(Q)$ and $\mathbf{D}^0(Q)$ constitute a partition of the set $\mathbf{D}$. Let us now assume that each diagnosis in $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ is assigned to its respective set in $\mathfrak{P}(Q)$ as per Definition 7.2 yielding the tuple $\langle \mathbf{D}_m^+(Q), \mathbf{D}_m^-(Q), \mathbf{D}_m^0(Q) \rangle$ where $\mathbf{D}_m^+(Q) \cup \mathbf{D}_m^-(Q) \cup \mathbf{D}_m^0(Q) = \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$. Then, by analogue argumentation as in the proof of Proposition 7.1, we obtain that $\mathbf{D}_m^+(Q)$, $\mathbf{D}_m^-(Q)$ and $\mathbf{D}_m^0(Q)$ are pairwise disjoint sets. That is, $\langle \mathbf{D}_m^+(Q), \mathbf{D}_m^-(Q), \mathbf{D}_m^0(Q) \rangle$ is the (extended) q-partition of Q w.r.t. the leading diagnoses set $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$.

By Remark 7.4, we have that $\mathbf{D}_{pos} := \mathbf{D}_m^+(Q) \cup \mathbf{D}_m^0(Q)$ are minimal diagnoses w.r.t. the DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P' \cup \{Q\}, N \cup N' \rangle_R$ (positive answer $u(Q)$) and $\mathbf{D}_{neg} := \mathbf{D}_m^-(Q) \cup \mathbf{D}_m^0(Q)$ are minimal diagnoses w.r.t. the DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \cup \{Q\} \rangle_R$ (negative answer $u(Q)$). Since $\mathbf{D}_{pos} \cup \mathbf{D}_{neg} \supseteq \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$, we have that each diagnosis in $\mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ is either in $\mathbf{D}_{pos}$ or in $\mathbf{D}_{neg}$ (or in both). Hence, for each diagnosis $\mathcal{D} \in \mathbf{mD}_{\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R}$ there is some answer $u(Q) \in \{true, false\}$ to the query Q such that $\mathcal{D}$ is a diagnosis w.r.t. the DPI resulting from $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle_R$ by addition of the new test case Q to the respective set ($P \cup P'$ for positive and $N \cup N'$ for negative answer). Consequently, the claimed proposition holds. $\square$

Corollary 13.1. *Algorithm 5 with $mode \in \{static, dynamic\}$ is unbiased for any given input DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle_R$.*

	STATICHS	DYNAMICHS
is used to solve	Interactive Static KB Debugging problem (Problem Definition 6.2)	Interactive Dynamic KB Debugging problem (Problem Definition 6.1)
soundness	yes	yes
completeness	yes	yes
optimality	yes	yes
number of solutions that must be considered (but not necessarily computed)	• initially fixed • upper bound: $\|\mathbf{mD}_{inputDPI}\|$	• not initially fixed, depends on specified test cases (answered queries) • upper bound: $\|\mathbf{aD}_{inputDPI}\|$
diagnoses	considers only minimal diagnoses w.r.t. the input DPI which satisfy all answered queries added as test cases so far	considers only minimal diagnoses w.r.t. the current DPI
conflict sets	computes only minimal conflict sets w.r.t. the input DPI	computes minimal conflict sets w.r.t. the current DPI
computes	a set $\mathbf{D}$ including the $\|\mathbf{D}\| \le n_{\max}$ (a-priori) most probable minimal diagnoses w.r.t. the input DPI which satisfy all answered queries added as test cases so far	a set $\mathbf{D}$ including the $\|\mathbf{D}\| \le n_{\max}$ (a-priori) most probable minimal diagnoses w.r.t. the current DPI
purpose of test cases	differentiation between minimal diagnoses of fixed DPI	obtaining a new DPI with fewer minimal diagnoses
set of all minimal diagnoses upon addition of a test case	is reduced to a proper subset	some are invalidated, some new ones might be introduced
set of all diagnoses upon addition of a test case	is reduced to a proper subset	is reduced to a proper subset
set of all minimal conflict sets upon addition of a test case	constant	some minimal conflict sets are reduced to smaller sets and/or some new minimal conflict sets (in no subset-relation with existing ones) are introduced
constructed tree: comparison to non-interactive wpHS-tree (Alg. 2)	equivalent (except for labels of leaf nodes)	might differ significantly
non-leaf-node labels	only minimal conflict sets w.r.t. the input DPI	(not necessarily minimal) conflict sets w.r.t. the current DPI
non-minimal and duplicate tree paths	deleted	stored
evolution of produced tree	only expansion (except for deletion of non-minimal and duplicate tree paths)	alternating tree expansion and pruning phases
pre-pruning (deletion of partial diagnoses)	only duplicate tree paths	any
post-pruning (deletion of complete diagnoses)	only non-minimal diagnoses (all invalidated minimal diagnoses are stored)	any
overall tree pruning	poor	significant
tree construction: worst case time and space complexity	• independent of specified test cases • upper and lower bound is time and space required by non-interactive wpHS-tree (Alg. 2)	• a function of the specified test cases and the leading diagnosis computation parameters $n_{\min}, n_{\max}, t$ • best case: significant savings compared to STATICHS • worst case: significant overhead compared to STATICHS
query generation	w.r.t. the current DPI	w.r.t. the current DPI
unbiased	yes	yes

Table 13.1: Comparison: STATICHS versus DYNAMICHS.

Part IV

Two Query Strategies for Efficient Fault Localization in Interactive Ontology Debugging

In this part, we suggest and extensively analyze different methods for the selection of an "optimal" query. The material dealt with in Part IV is based on the publications [SFFR12, SF10] where the former was published in the journal *Web Semantics: Science, Services and Agents on the World Wide Web* and the latter in the *Proceedings of the 9th International Semantic Web Conference (ISWC 2010).*

Chapter 14

Introduction to the Problem

Ontology acquisition and maintenance are important prerequisites for the successful application of semantic systems in areas such as the Semantic Web. However, as state of the art ontology extraction methods cannot automatically acquire ontologies in a complete and error-free fashion, users of such systems must formulate and correct logical descriptions on their own. In most of the cases these users are domain experts who have little or no experience in expressing knowledge in representation languages like OWL 2 DL [GHM⁺08]. Studies in cognitive psychology, e.g. [CP71, JL99], indicate that humans make systematic errors while formulating or interpreting logical descriptions, with the results presented in [RDH⁺04, RCVB09] confirming that these observations also apply to ontology development. Moreover, the problem gets even more if an ontology is developed by a group of users, such as OBO Foundry[29] or NCI Thesaurus[30], is based on a set of imported third-party ontologies, etc. In this case inconsistencies might appear if some user does not understand or accept the *context* in which shared ontological descriptions are used. Therefore, identification of erroneous ontological definitions is a difficult and time-consuming task.

Several ontology debugging methods [SHCH07, KPHS07, FS05, HPS08] were proposed to simplify ontology development and maintenance. Usually the main aim of debugging is to obtain a consistent and, optionally, coherent ontology. These basic requirements can be extended with additional ones, such as test cases [FS05], which must be fulfilled by the *target ontology* $\mathcal{O}_t$. Any ontology that does not fulfill the requirements is *faulty* regardless of how it was created. For instance, an ontology might be created by an expert specializing descriptions of the imported ontologies (top-down) or by an inductive learning algorithm from a set of examples (bottom-up).

Note that even if all requirements are completely specified, many logically equivalent target ontologies might exist. They may differ in aspects such as the complexity of consistency checks, size or readability. However, selecting between logically equivalent theories based on such measures is out of the scope of this work. Furthermore, although target ontologies may evolve as requirements change over time, we assume that the target ontology remains stable throughout a debugging session.

Given an set of requirements (e.g. formulated by a user) and a faulty ontology, the task of an ontology debugger is to identify the set of alternative diagnoses, where each diagnosis corresponds to a set of possibly faulty axioms. More concretely, a *diagnosis* $\mathcal{D}$ is a subset of an ontology $\mathcal{O}$ such that one should remove (change) all the axioms of a diagnosis from the ontology (i.e. $\mathcal{O} \setminus \mathcal{D}$) in order to formulate an ontology $\mathcal{O}'$ that fulfills all the given requirements. Only if the set of requirements is complete the only possible ontology $\mathcal{O}'$ corresponds to the target ontology $\mathcal{O}_t$. In the following we refer to the removal of a diagnosis from the ontology as a *trivial application* of a diagnosis. Moreover, in practical applica-

---

[29]http://www.obofoundry.org
[30]http://ncit.nci.nih.gov

tions it might be inefficient to consider all possible diagnoses. Therefore, modern ontology debugging approaches focus on the computation of minimal diagnoses. A set of axioms $\mathcal{D}_i$ is a *minimal diagnosis* iff there is no proper subset $\mathcal{D}_i' \subset \mathcal{D}_i$ which is a diagnosis. Thus, minimal diagnoses constitute minimal required changes to the ontology.

Application of diagnosis methods can be problematic in the cases for which many alternative minimal diagnoses exist for a given set of test cases and requirements. A sample study of real-world incoherent ontologies, which were used in [KPHS07], showed that hundreds or even thousands of minimal diagnoses may exist. In the case of the Transportation ontology the diagnosis method was able to identify 1782 minimal diagnoses [31]. In such situations a simple visualization of all alternative sets of modifications to the ontology is ineffective. Thus an efficient debugging method should be able to discriminate between the diagnoses in order to select the *target diagnosis* $\mathcal{D}_t$. Trivial application of $\mathcal{D}_t$ to the ontology $\mathcal{O}$ allows a user to extend $(\mathcal{O} \setminus \mathcal{D}_t)$ with a set of additional axioms EX and, thus, to formulate the target ontology $\mathcal{O}_t$, i.e. $\mathcal{O}_t = (\mathcal{O} \setminus \mathcal{D}_t) \cup EX$.

One possible solution to the diagnosis discrimination problem would be to order the set of diagnoses by various preference criteria. For instance, Kalyanpur et al. [KPSCG06] suggest a measure to rank the axioms of a diagnosis depending on their structure, usage in test cases, provenance, and impact in terms of entailments. Only the top ranking diagnoses are then presented to the user. Of course this set of diagnoses will contain the target diagnosis only in cases where the faulty ontology, the given requirements and test cases provide sufficient data to the appropriate heuristic. However, it is difficult to identify which information, e.g. test cases, is really required to identify the target diagnosis. That is, a user does not know a priori which and how many tests should be provided to the debugger to ensure that it will return the target diagnosis.

In this part we present an approach for the acquisition of additional information by *generating* a sequence of queries, the answers of which can be used to reduce the set of diagnoses and ultimately identify the target diagnosis. These queries should be answered by an oracle such as a user or an information extraction system. In order to construct queries we exploit the property that different ontologies resulting from trivial applications of different diagnoses entail unequal sets of axioms. Consequently, we can differentiate between diagnoses by asking the oracle if the target ontology should entail a set of logical sentences or not. These entailed logical sentences can be generated by the classification and realization services provided in description logic reasoning systems [SPG+07, HM01, MSH09]. In particular, the classification process computes a subsumption hierarchy (sometimes also called "inheritance hierarchy" of parents and children) for each concept description mentioned in a TBox. For each individual mentioned in an ABox, the realization computes all the concept names of which the individual is an instance [SPG+07].

We propose two methods for selecting the next query of the set of possible queries: The first method employs a greedy approach that selects queries which try to cut the number of diagnoses in half. The second method exploits the fact that some diagnoses are more likely than others because of typical user errors [RDH+04, RCVB09]. Beliefs for an error to occur in a given part of a knowledge base, represented as a probability, can be used to estimate the change in entropy of the set of diagnoses if a particular query is answered. In our evaluation the fault probabilities of axioms are estimated by the type and number of the logical operators employed. For example, roughly speaking, the greater the number of logical operators and the more complex these operators are, the greater the fault probability of an axiom. For assigning prior fault probabilities to diagnoses we employ the fault probabilities of axioms. Of course other methods for guessing prior fault probabilities, e.g. based on context of concept descriptions, measures suggested in the previous work [KPSCG06], etc., can be easily integrated in our framework. Given a set of diagnoses and their probabilities the method selects a query which minimizes the expected entropy of a set of diagnoses after an oracle answers a query, i.e. maximizes the information gain. An oracle should answer such queries until a diagnosis is identified whose probability is significantly higher than those of all other

---

[31]In Chapter 18, we will give a detailed characterization of these ontologies.

diagnoses. This diagnosis is most likely to be the target diagnosis.

In the first evaluation scenario we compare the performance of both methods in terms of the number of queries needed to identify the target diagnosis. The evaluation is performed using generated examples as well as real-world ontologies presented in Tables 18.1 and 18.5. In the first case we alter a consistent and coherent ontology with additional axioms to generate conflicts that result in a predefined number of diagnoses of a required length. Each faulty ontology is then analyzed by the debugging algorithm using entropy, greedy and "random" strategies, where the latter selects queries at random. The evaluation results show that in some cases the entropy-based approach is almost 60% better than the greedy one whereas both approaches clearly outperformed the random strategy.

In the second evaluation scenario we investigate the robustness of the entropy-based strategy with respect to variations in the prior fault probabilities. We analyze the performance of entropy-based and greedy strategies on real-world ontologies by simulating different types of prior fault probability distributions as well as the "quality" of these probabilities that might occur in practice. In particular, we identify the cases where all prior fault probabilities are (1) equal, (2) "moderately" varied or (3) "extremely" varied. Regarding the "quality" of the probabilities we investigate cases where the guesses based on the prior diagnosis probabilities are good, average or bad. The results show that the entropy method outperforms "split-in-half" in almost all of the cases, namely when the target diagnosis is located in the more likely two thirds of the minimal diagnoses. In some situations the entropy-based approach achieves even twice the performance of the greedy one. Only in cases where the initial guess of the prior probabilities is very vague (the bad case), *and* the number of queries needed to identify the target diagnosis is low, "split-in-half" may save on average one query. However, if the number of queries increases, the performance of the entropy-based query selection increases compared to the "split-in-half" strategy. We observed that if the number of queries is greater than 10, the entropy-based method is preferable even if the initial guess of the prior probabilities is bad. This is due to the effect that the initial bad guesses are improved by the Bayes-update of the diagnoses probabilities as well as an ability of the entropy-based method to stop in the cases when a probability of some diagnosis is above an acceptance threshold predefined by the user. Consequently, entropy-based query selection is robust enough to handle different prior fault probability distributions.

Additional experiments performed on big real-world ontologies demonstrate the scalability of the suggested approach. In our experiments we were able to identify the target diagnosis in an ontology with over 33000 axioms using entropy-based query selection in only 190 seconds using an average of five queries.

The remainder of Part IV is organized as follows: Chapter 15 presents two introductory examples as well as the basic concepts. The details of the entropy-based query selection method are given in Chapter 16. Chapter 17 describes the implementation of the approach and is followed by evaluation results in Chapter 18. An overview of related work is given in Chapter 19 and conclusions are drawn in Chapter 20.

Chapter 15

Motivating Examples and Basic Concepts

We begin by presenting the fundamentals of ontology diagnosis and then show how queries and answers can be generated and employed to differentiate between sets of diagnoses.

Description Logics

Since the underlying knowledge representation method of ontologies in the Semantic Web is based on description logics, we start by briefly introducing the main concepts, employing the usual definitions as in [Bor96, Baa03]. A knowledge base is comprised of two components, namely a TBox (denoted by $\mathcal{T}$) and a ABox ($\mathcal{A}$). The TBox defines the terminology whereas the ABox contains assertions about named individuals in terms of the vocabulary defined in the TBox. The vocabulary consists of concepts, denoting sets of individuals, and roles, denoting binary relationships between individuals. These concepts and roles may be either atomic or complex, the latter being obtained by employing description operators. The language of descriptions is defined recursively by starting from a schema $S = (\mathcal{CN}, \mathcal{RN}, \mathcal{IN})$ of disjoint sets of names for concepts, roles, and individuals. Typical operators for the construction of complex descriptions are $C \sqcup D$ (disjunction), $C \sqcap D$ (conjunction), $\neg C$ (negation), $\forall R.C$ (concept value restriction), and $\exists R.C$ (concept exists restriction), where C and D are elements of $\mathcal{CN}$ and $R \in \mathcal{RN}$.

Knowledge bases are defined by a finite set of logical sentences. Sentences regarding the TBox are called terminological axioms whereas sentences regarding the ABox are called assertional axioms. Terminological axioms are expressed by $C \sqsubseteq D$ (Generalized Concept Inclusion) which corresponds to the logical implication. Let $a, b \in \mathcal{IN}$ be individual names. $C(a)$ and $R(a, b)$ are thus assertional axioms.

Concepts (rsp. roles) can be regarded as unary (rsp. binary) predicates. Roughly speaking description logics can be seen as fragments of first-order predicate logic (without considering transitive closure or special fixpoint semantics). These fragments are specifically designed to ensure decidability or favorable computational costs.

The semantics of description terms are usually given using an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, (\cdot)^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is a domain (non-empty universe) of values, and $(\cdot)^{\mathcal{I}}$ is a function that maps every concept description to a subset of $\Delta^{\mathcal{I}}$, and every role name to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The mapping also associates a value in $\Delta^{\mathcal{I}}$ with every individual name in $\mathcal{IN}$. An interpretation $\mathcal{I}$ is a model of a knowledge base iff it satisfies all terminological axioms and assertional axioms. A knowledge base is satisfiable iff a model exists. A concept description C is coherent (satisfiable) w.r.t. a TBox $\mathcal{T}$, if a model $\mathcal{I}$ of $\mathcal{T}$ exists such that $C^{\mathcal{I}} \neq \emptyset$.

A TBox is incoherent iff an incoherent concept description exists.

Diagnosis of Ontologies

Example 15.1   Consider a simple ontology $\mathcal{O}$ with the terminology $\mathcal{T}$:

$$ax_1 : A \sqsubseteq B \quad ax_2 : B \sqsubseteq C$$
$$ax_3 : C \sqsubseteq D \quad ax_4 : D \sqsubseteq R$$

and assertions $\mathcal{A} : \{A(w), \neg R(w), A(v)\}$.

Assume that the user explicitly states that the three assertional axioms should be considered as correct, i.e. these axioms are added to a background theory $\mathcal{B}$. The introduction of a background theory ensures that the diagnosis method focuses purely on the potentially faulty axioms.

Furthermore, assume that the user requires the currently inconsistent ontology $\mathcal{O} \cup \mathcal{B}$ to be consistent. The only irreducible set of non-background axioms (minimal conflict set) that preserves the inconsistency is $CS : \langle ax_1, ax_2, ax_3, ax_4 \rangle$. That is, one has to modify or remove the axioms of at least one of the following diagnoses

$$\mathcal{D}_1 : [ax_1] \quad \mathcal{D}_2 : [ax_2] \quad \mathcal{D}_3 : [ax_3] \quad \mathcal{D}_4 : [ax_4]$$

to restore the consistency of the ontology. However, it is unclear which of the ontologies $\mathcal{O}_i = \mathcal{O} \setminus \mathcal{D}_i$ obtained by application of diagnoses from the set $\mathbf{D} : \{\mathcal{D}_1, \ldots, \mathcal{D}_4\}$ is the target one.   □

Definition 15.1. *A target ontology $\mathcal{O}_t$ is a set of logical sentences characterized by a set of background axioms $\mathcal{B}$, a set of sets of logical sentences P that must be entailed by $\mathcal{O}_t$ and the set of sets of logical sentences N that must not be entailed by $\mathcal{O}_t$.*

A target ontology $\mathcal{O}_t$ must fulfill the following necessary requirements:

- *$\mathcal{O}_t$ must be satisfiable (optionally coherent)*

- *$\mathcal{B} \subseteq \mathcal{O}_t$*

- *$\mathcal{O}_t \models p \quad \forall p \in P$*

- *$\mathcal{O}_t \not\models n \quad \forall n \in N$*

Given $\mathcal{B}$, P, and N, an ontology $\mathcal{O}$ is faulty iff $\mathcal{O}$ does not fulfill all the necessary requirements of the target ontology.

Note that the approach presented in this work can be used with any knowledge representation language for which there exists a sound and complete procedure to decide whether $\mathcal{O} \models ax$ and the entailment operator $\models$ is extensive, monotone and idempotent. For instance, these requirements are fulfilled by all subsets of OWL 2 which are interpreted under OWL Direct Semantics.

Definition 15.1 allows a user to identify the target diagnosis $\mathcal{D}_t$ by providing sufficient information about the target ontology in the sets $\mathcal{B}$, P and N. For instance, if in Example 15.1 the user provides the information that $\mathcal{O}_t \models \{B(w)\}$ and $\mathcal{O}_t \not\models \{C(w)\}$, the debugger will return only one diagnosis, namely $\mathcal{D}_2$. Application of this diagnosis results in a consistent ontology $\mathcal{O}_2 = \mathcal{O} \setminus \mathcal{D}_2$ that – integrated with the background knowledge $\mathcal{B}$ – entails $\{B(w)\}$ because of ax_1 and the assertion $A(w)$. In addition, $\mathcal{O}_2 \cup \mathcal{B}$ does not entail $\{C(w)\}$ since $\mathcal{O}_2 \cup \mathcal{B} \cup \{\neg C(w)\}$ is consistent and, moreover, $\{\neg R(w), ax_4, ax_3\} \models \{\neg C(w)\}$. All other ontologies $\mathcal{O}_i = (\mathcal{O} \setminus \mathcal{D}_i)$ obtained by the application of the diagnoses $\mathcal{D}_1, \mathcal{D}_3$ and $\mathcal{D}_4$ do not fulfill the given requirements, since $\mathcal{O}_1 \cup \mathcal{B} \cup \{B(w)\}$ is inconsistent and therefore any consistent extension of $\mathcal{O}_1 \cup \mathcal{B}$ cannot entail $\{B(w)\}$. As both $\mathcal{O}_3 \cup \mathcal{B}$ and $\mathcal{O}_4 \cup \mathcal{B}$ entail $\{C(w)\}$, $\mathcal{O}_2 \cup \mathcal{B}$ corresponds to the target ontology $\mathcal{O}_t$.

Definition 15.2. *Let* $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ *be a diagnosis problem instance, where* $\mathcal{O}$ *is an ontology,* $\mathcal{B}$ *a background theory,* P *a set of sets of logical sentences which must be entailed by the target ontology* $\mathcal{O}_t$*, and* N *a set of sets of logical sentences which must* not *be entailed by* $\mathcal{O}_t$.

A set of axioms $\mathcal{D} \subseteq \mathcal{O}$ *is a diagnosis iff the set of axioms* $\mathcal{O} \setminus \mathcal{D}$ *can be extended by a logical description* EX *such that:*

1. $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX$ *is consistent (and coherent if required)*

2. $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \models p \quad \forall p \in P$

3. $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \not\models n \quad \forall n \in N$

A diagnosis $\mathcal{D}_i$ defines a partition of the ontology $\mathcal{O}$ where each axiom $ax_j \in \mathcal{D}_i$ is a candidate for changes by the user and each axiom $ax_k \in \mathcal{O} \setminus \mathcal{D}_i$ is correct. If $\mathcal{D}_t$ is the set of axioms of $\mathcal{O}$ to be changed (i.e. $\mathcal{D}_t$ is the target diagnosis) then the target ontology $\mathcal{O}_t$ is $(\mathcal{O} \setminus \mathcal{D}_t) \cup \mathcal{B} \cup EX$ for some EX defined by the user.

In the following we assume the background theory $\mathcal{B}$ together with the sets of logical sentences in the sets P and N always allow formulation of the target ontology. Moreover, a diagnosis exists iff a target ontology exists.

Proposition 15.1. *A diagnosis* $\mathcal{D}$ *for a diagnosis problem instance* $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ *exists iff*

$$\mathcal{B} \cup \bigcup_{p \in P} p$$

is consistent (coherent) and

$$\forall n \in N \; : \; \mathcal{B} \cup \bigcup_{p \in P} p \not\models n$$

The set of all diagnoses is complete in the sense that at least one diagnosis exists where the ontology resulting from the trivial application of a diagnosis is a subset of the target ontology:

Proposition 15.2. *Let* $\mathbf{D} \neq \emptyset$ *be the set of all diagnoses for a diagnosis problem instance* $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ *and* $\mathcal{O}_t$ *the target ontology. Then a diagnosis* $\mathcal{D}_t \in \mathbf{D}$ *exists s.t.* $(\mathcal{O} \setminus \mathcal{D}_t) \subseteq \mathcal{O}_t$.

The set of all diagnoses can be characterized by the set of minimal diagnoses.

Definition 15.3. *A diagnosis* $\mathcal{D}$ *for a diagnosis problem instance* $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ *is a* minimal diagnosis *iff there is no* $\mathcal{D}' \subset \mathcal{D}$ *such that* $\mathcal{D}'$ *is a diagnosis.*

Proposition 15.3. *Let* $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ *be a diagnosis problem instance. For every diagnosis* $\mathcal{D}$ *there is a minimal diagnosis* $\mathcal{D}'$ *s.t.* $\mathcal{D}' \subseteq \mathcal{D}$.

Definition 15.4. *A diagnosis* $\mathcal{D}$ *for a diagnosis problem instance* $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ *is a* minimum cardinality diagnosis *iff there is no diagnosis* $\mathcal{D}'$ *such that* $|\mathcal{D}'| < |\mathcal{D}|$.

To summarize, a diagnosis describes which axioms are candidates for modification. Despite the fact that multiple diagnoses may exist, some are more preferable than others. E.g. minimal diagnoses require minimal changes, i.e. axioms are not considered for modification unless there is a reason. Minimal cardinality diagnoses require changing a minimal number of axioms. The actual type of error contained in an axiom is irrelevant as the concept of diagnosis defined here does not make any assumptions about errors themselves. There can, however, be instances where an ontology is faulty and the empty diagnosis is the only minimal diagnosis, e.g. if some axioms are missing and nothing must be changed.

The extension EX plays an important role in the ontology repair process, suggesting axioms that should be added to the ontology. For instance, in Example 15.1 the user requires that the target ontology

must not entail $\{B(w)\}$ but has to entail $\{B(v)\}$, that is $N = \{\{B(w)\}\}$ and $P = \{\{B(v)\}\}$. Because, the example ontology $\mathcal{O}$ is inconsistent some sentences must be changed. The consistent ontology $\mathcal{O}_1 = \mathcal{O} \setminus \mathcal{D}_1$ (along with the background axioms $\mathcal{B}$) neither entails $\{B(v)\}$ nor $\{B(w)\}$ (in particular $\mathcal{O}_1 \cup \mathcal{B} \models \{\neg B(w)\}$). Consequently, $\mathcal{O}_1$ has to be extended with a set EX of logical sentences in order to entail $\{B(v)\}$. This set of logical sentences can be approximated with $EX = \{B(v)\}$. $\mathcal{O}_1 \cup \mathcal{B} \cup EX$ is satisfiable, entails $\{B(v)\}$ but does not entail $\{B(w)\}$. All other ontologies $O_i = \mathcal{O} \setminus \mathcal{D}_i$, $i = 2, 3, 4$ (integrated with $\mathcal{B}$) are consistent but entail $\{B(w), B(v)\}$ and must be rejected because of the monotonic semantics of description logic. That is, there is no such extension EX that $(\mathcal{O}_i \cup \mathcal{B} \cup EX) \not\models \{B(w)\}$. Therefore, the diagnosis $\mathcal{D}_1$ is the minimum cardinality diagnosis which allows the formulation of the target ontology. Note that formulation of the complete extension is impossible, since our diagnosis approach deals with changes to existing axioms and does not learn new axioms.

The following corollary characterizes diagnoses without employing the true extension EX to formulate the target ontology. The idea is to use the sentences which must be entailed by the target ontology to approximate EX as shown above.

Corollary 15.1. *Given a diagnosis problem instance $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$, a set of axioms $\mathcal{D} \subseteq \mathcal{O}$ is a diagnosis iff*

$$(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup \bigcup_{p \in P} p \qquad \text{(Condition 1)}$$

is satisfiable (coherent) and

$$\forall n \in N \; : \; (\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup \bigcup_{p \in P} p \not\models n \qquad \text{(Condition 2)}$$

Proof sketch: ($\Rightarrow$) Let $\mathcal{D} \subseteq \mathcal{O}$ be a diagnosis for $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$. Since there is an EX s.t. $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX$ is satisfiable (coherent) and $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \models p$ for all $p \in P$, it follows that $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \cup \bigcup_{p \in P} p$ is satisfiable (coherent) and therefore $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup \bigcup_{p \in P} p$ is satisfiable (coherent). Consequently, the first condition of the corollary is fulfilled. Since $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \models p$ for all $p \in P$ and $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \not\models n$ for all $n \in N$ it follows that $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \cup \bigcup_{p \in P} p \not\models n$ for all $n \in N$. Consequently, $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup \bigcup_{p \in P} p \not\models n$ for all $n \in N$ and the second condition of the corollary is fulfilled.

($\Leftarrow$) Let $\mathcal{D} \subseteq \mathcal{O}$ and $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ be a diagnosis problem instance. Without limiting generality let $EX = P$. By Condition 1 of the corollary $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup \bigcup_{p \in P} p$ is satisfiable (coherent). Therefore, for $EX = P$ the sentences $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX$ are satisfiable (coherent), i.e. the first condition for a diagnosis is fulfilled and these sentences entail p for all $p \in P$ which corresponds to the second condition a diagnosis must fulfill. Furthermore, by Condition 2 of the corollary $(\mathcal{O} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \not\models n$ for all $n \in N$ holds and therefore the third condition for a diagnosis is fulfilled. Consequently, $\mathcal{D} \subseteq \mathcal{O}$ is a diagnosis for $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$. $\qquad\square$

Conflict sets, which are the parts of the ontology that preserve the inconsistency/incoherency, are usually employed to constrain the search space during computation of diagnoses.

Definition 15.5. *Given a diagnosis problem instance $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$, a set of axioms $CS \subseteq \mathcal{O}$ is a conflict set iff $CS \cup \mathcal{B} \cup \bigcup_{p \in P} p$ is inconsistent (incoherent) or $n \in N$ exists s.t. $CS \cup \mathcal{B} \cup \bigcup_{p \in P} p \models n$.*

Definition 15.6. *A conflict set CS for an instance $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ is minimal iff there is no $CS' \subset CS$ such that CS' is a conflict set.*

A set of minimal conflict sets can be used to compute the set of minimal diagnoses as shown in [Rei87]. The idea is that each diagnosis must include at least one element of each minimal conflict set.

Proposition 15.4. *$\mathcal{D}$ is a minimal diagnosis for the diagnosis problem instance $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ iff $\mathcal{D}$ is a minimal hitting set for the set of all minimal conflict sets of $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$.*

Ontology	Entailments
$\mathcal{O}_1$	$\emptyset$
$\mathcal{O}_2$	$\{B(w)\}$
$\mathcal{O}_3$	$\{B(w), C(w)\}$
$\mathcal{O}_4$	$\{B(w), C(w), D(w)\}$

Table 15.1: Entailments of ontologies $\mathcal{O}_i = (\mathcal{O} \setminus \mathcal{D}_i)$, $i = 1, \dots, 4$ (integrated with $\mathcal{B}$) in Example 15.1 returned by realization.

Given a set of sets $\overline{S}$, a set H is a hitting set of $\overline{S}$ iff $H \cap S_i \neq \emptyset$ for all $S_i \in \overline{S}$ and $H \subseteq \bigcup_{S_i \in \overline{S}} S_i$. Most modern ontology diagnosis methods [SHCH07, KPHS07, FS05, HPS08] are implemented according to Proposition 28.2 and differ only in details, such as how and when (minimal) conflict sets are computed, the order in which hitting sets are generated, etc.

Differentiating between Diagnoses

The diagnosis method usually generates a set of diagnoses for a given diagnosis problem instance. Thus, in Example 15.1 an ontology debugger returns a set of four minimal diagnoses $\{\mathcal{D}_1, \dots, \mathcal{D}_4\}$. As explained in the previous section, additional information, i.e. sets of sets of logical sentences P and N, can be used by the debugger to reduce the set of diagnoses. However, in the general case the user does not know which sets P and N to provide to the debugger such that the target diagnosis will be identified. Therefore, the debugger should be able to identify sets of logical sentences on its own and only ask the user or some other oracle, whether these sentences *must* or *must not* be entailed by the target ontology. To generate these sentences the debugger can apply each of the diagnoses in $\mathbf{D} = \{\mathcal{D}_1, \dots, \mathcal{D}_n\}$ and obtain a set of ontologies $\mathcal{O}_i = \mathcal{O} \setminus \mathcal{D}_i$, $i = 1, \dots, n$ that fulfill the user requirements. For each ontology $\mathcal{O}_i$ a description logic reasoner can generate a set of entailments such as entailed subsumptions provided by the classification service and sets of class assertions provided by the realization service. These entailments can be used to discriminate between the diagnoses, as different ontologies entail different sets of sentences due to extensivity of the entailment relation. Note that in the examples provided in this section we consider only two types of entailments, namely subsumption and class assertion. In general, the approach presented in this work is not limited to these types and can use all of the entailment types supported by a reasoner.

For instance, in Example 15.1 for each ontology $\mathcal{O}_i = (\mathcal{O} \setminus \mathcal{D}_i)$, $i = 1 \dots 4$ (integrated with $\mathcal{B}$) the realization service of a reasoner returns the set of class assertions presented in Table 15.1. Without any additional information the debugger cannot decide which of these sentences must be entailed by the target ontology. To obtain this information the diagnosis method must query an oracle that can specify whether the target ontology entails some set of sentences or not. E.g. the debugger could ask an oracle if $\{D(w)\}$ is entailed by the target ontology ($\mathcal{O}_t \models \{D(w)\}$). If the answer is *yes*, then $\{D(w)\}$ is added to P and $\mathcal{D}_4$ is considered as the target diagnosis. All other diagnoses are rejected because $(\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \{D(w)\}$ for $i = 1, 2, 3$ is inconsistent. If the answer is *no*, then $\{D(w)\}$ is added to N and $\mathcal{D}_4$ is rejected as $(\mathcal{O} \setminus \mathcal{D}_4) \cup \mathcal{B} \models \{D(w)\}$ and we have to ask the oracle another question. In the following we consider a query Q as a set of logical sentences such that $\mathcal{O}_t \models Q$ holds iff $\mathcal{O}_t \models q_i$ for all $q_i \in Q$.

Property 1. *Given a diagnosis problem instance $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$, a set of diagnoses $\mathbf{D}$, a set of logical sentences Q representing the query $(\mathcal{O}_t \models Q)$ and an oracle able to evaluate the query:*

If the oracle answers yes *then every diagnosis $\mathcal{D}_i \in \mathbf{D}$ is a diagnosis for $P \cup \{Q\}$ iff both conditions*

hold:

$$(\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \bigcup_{p \in P} p \cup Q \text{ is consistent (coherent)}$$

$$\forall n \in N \; : \; (\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \bigcup_{p \in P} p \cup Q \not\models n$$

If the oracle answers no *then every diagnosis* $\mathcal{D}_i \in \mathbf{D}$ *is a diagnosis for* $N \cup \{Q\}$ *iff both conditions hold:*

$$(\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \bigcup_{p \in P} p \text{ is consistent (coherent)}$$

$$\forall n \in (N \cup \{Q\}) \; : \; (\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \bigcup_{p \in P} p \not\models n$$

In particular, a query partitions the set of diagnoses $\mathbf{D}$ into three disjoint subsets.

Definition 15.7. *For a query Q, each diagnosis $\mathcal{D}_i \in \mathbf{D}$ of a diagnosis problem instance $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ can be assigned to one of the three sets $\mathbf{D^P}$, $\mathbf{D^N}$ or $\mathbf{D}^\emptyset$ where*

- $\mathcal{D}_i \in \mathbf{D^P}$ *iff it holds that*

$$(\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \bigcup_{p \in P} p \models Q$$

- $\mathcal{D}_i \in \mathbf{D^N}$ *iff it holds that*

$$(\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \bigcup_{p \in P} p \cup Q$$

is inconsistent (incoherent).

- $\mathcal{D}_i \in \mathbf{D}^\emptyset$ *iff* $\mathcal{D}_i \in \mathbf{D} \setminus \left(\mathbf{D^P} \cup \mathbf{D^N} \right)$

Given a diagnosis problem instance we say that the diagnoses in $\mathbf{D^P}$ predict a positive answer (*yes*) as a result of the query Q, diagnoses in $\mathbf{D^N}$ predict a negative answer (*no*), and diagnoses in $\mathbf{D}^\emptyset$ do not make any predictions.

Property 2. *Given a diagnosis problem instance $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$, a set of diagnoses $\mathbf{D}$, a query Q and an oracle:*

If the oracle answers yes *then the set of rejected diagnoses is $\mathbf{D^N}$ and the set of remaining diagnoses is $\mathbf{D^P} \cup \mathbf{D}^\emptyset$.*

If the oracle answers no *then the set of rejected diagnoses is $\mathbf{D^P}$ and the set of remaining diagnoses is $\mathbf{D^N} \cup \mathbf{D}^\emptyset$.*

Consequently, given a query Q either $\mathbf{D^P}$ or $\mathbf{D^N}$ is eliminated but $\mathbf{D}^\emptyset$ always remains after the query is answered. For generating queries we have to investigate for which subsets $\mathbf{D^P}, \mathbf{D^N} \subseteq \mathbf{D}$ a query exists that can differentiate between these sets. A straight forward approach is to investigate all possible subsets of $\mathbf{D}$. In our evaluation we show that this is feasible if we limit the number n of minimal diagnoses to be considered during query generation and selection. E.g. for $n = 9$, the algorithm has to verify 512 possible partitions in the worst case.

Given a set of diagnoses $\mathbf{D}$ for the ontology $\mathcal{O}$, a set P of sets of sentences that must be entailed by the target ontology $\mathcal{O}_t$ and a set of background axioms $\mathcal{B}$, the set of partitions $\mathbf{PR}$ for which a query exists can be computed as follows:

1. Generate the power set $\mathcal{P}(\mathbf{D})$, $\mathbf{PR} \leftarrow \emptyset$

2. Assign an element of $\mathcal{P}(\mathbf{D})$ to the set $\mathbf{D_i^P}$ and generate a set of common entailments E_i of all ontologies $(\mathcal{O} \setminus \mathcal{D}_j) \cup \mathcal{B} \cup \bigcup_{p \in P} p$, where $\mathcal{D}_j \in \mathbf{D_i^P}$

3. If $E_i = \emptyset$, then reject the current element $\mathbf{D_i^P}$, i.e. set $\mathcal{P}(\mathbf{D}) \leftarrow \mathcal{P}(\mathbf{D}) \setminus \{\mathbf{D_i^P}\}$ and goto Step 2. Otherwise set $Q_i \leftarrow E_i$.

4. Use Definition 15.7 and the query Q_i to classify the diagnoses $\mathcal{D}_k \in \mathbf{D} \setminus \mathbf{D_i^P}$ into the sets $\mathbf{D_i^P}$, $\mathbf{D_i^N}$ and $\mathbf{D_i^\emptyset}$. The generated partition is added to the set of partitions $\mathbf{PR} \leftarrow \mathbf{PR} \cup \{\langle Q_i, \mathbf{D_i^P}, \mathbf{D_i^N}, \mathbf{D_i^\emptyset}\rangle\}$ and set $\mathcal{P}(\mathbf{D}) \leftarrow \mathcal{P}(\mathbf{D}) \setminus \{\mathbf{D_i^P}\}$. If $\mathcal{P}(\mathbf{D}) \neq \emptyset$ then go to Step 2.

In Example 15.1 the set of diagnoses $\mathbf{D}$ of the ontology $\mathcal{O}$ contains 4 elements. Therefore, the power set $\mathcal{P}(\mathbf{D})$ includes 15 elements $\{\{\mathcal{D}_1\}, \{\mathcal{D}_2\}, \ldots, \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}\}$, assuming we omit the element corresponding to $\emptyset$ as it does not contain any diagnoses to be evaluated. Moreover, assume that P and N are empty. In each iteration an element of $\mathcal{P}(\mathbf{D})$ is assigned to the set $\mathbf{D_i^P}$. For instance, the algorithm assigns $\mathbf{D_1^P} = \{\mathcal{D}_1, \mathcal{D}_2\}$. In this case the set of common entailments is empty as $(\mathcal{O} \setminus \mathcal{D}_1) \cup \mathcal{B}$ has no entailed sentences (see Table 15.1). Therefore, the set $\{\mathcal{D}_1, \mathcal{D}_2\}$ is rejected and removed from $\mathcal{P}(\mathbf{D})$. Assume that in the next iteration the algorithm selects $\mathbf{D_2^P} = \{\mathcal{D}_2, \mathcal{D}_3\}$. In this case the set of common entailments $E_2 = \{B(w)\}$ is not empty and so $Q_2 = \{B(w)\}$. The remaining diagnoses $\mathcal{D}_1$ and $\mathcal{D}_4$ are classified according to Definition 15.7. That is, the algorithm selects the first diagnosis $\mathcal{D}_1$ and verifies whether $(\mathcal{O} \setminus \mathcal{D}_1) \cup \mathcal{B} \models \{B(w)\}$. Given the negative answer of the reasoner, the algorithm checks if $(\mathcal{O} \setminus \mathcal{D}_1) \cup \mathcal{B} \cup \{B(w)\}$ is inconsistent. Since the condition is satisfied the diagnosis $\mathcal{D}_1$ is added to the set $\mathbf{D_2^N}$. The second diagnosis $\mathcal{D}_4$ is added to the set $\mathbf{D_2^P}$ as it satisfies the first requirement $(\mathcal{O} \setminus \mathcal{D}_4) \cup \mathcal{B} \models \{B(w)\}$. The resulting partition $\langle\{B(w)\}, \{\mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}, \{\mathcal{D}_1\}, \emptyset\rangle$ is added to the set $\mathbf{PR}$.

However, a query need not include all of the entailed sentences. If a query Q partitions the set of diagnoses into $\mathbf{D^P}$, $\mathbf{D^N}$ and $\mathbf{D^\emptyset}$ and an (irreducible) subset $Q' \subset Q$ exists which preserves the partition then it is sufficient to query Q'. In our example, $Q_2 : \{B(w), C(w)\}$ can be reduced to its subset $Q_2' : \{C(w)\}$. If there are multiple irreducible subsets that preserve the partition then we select one of them.

All of the queries and their corresponding partitions generated in Example 15.1 are presented in Table 15.2. Given these queries the debugger has to decide which one should be asked first in order to minimize the number of queries to be answered. A popular query selection heuristic (called "split-in-half") prefers queries which allow half of the diagnoses to be removed from the set $\mathbf{D}$ regardless of the answer of an oracle.

Using the data presented in Table 15.2, the "split-in-half" heuristic determines that asking the oracle if $(\mathcal{O}_t \models \{C(w)\})$ is the best query (i.e. the reduced query Q_2), as two diagnoses from the set $\mathbf{D}$ are removed regardless of the answer. Assuming that $\mathcal{D}_1$ is the target diagnosis, then an oracle will answer *no* to our question (i.e. $\mathcal{O}_t \not\models \{C(w)\}$). Based on this feedback, the diagnoses $\mathcal{D}_3$ and $\mathcal{D}_4$ are removed according to Property 2. Given the updated set of diagnoses $\mathbf{D}$ and $P = \{\{C(w)\}\}$ the partitioning algorithm returns the only partition $\langle\{B(w)\}, \{\mathcal{D}_2\}, \{\mathcal{D}_1\}, \emptyset\rangle$. The heuristic then selects the query $\{B(w)\}$, which is also answered with *no* by the oracle. Consequently, $\mathcal{D}_1$ is identified as the only remaining minimal diagnosis.

In general, if n is the number of diagnoses and we can split the set of diagnoses in half with each query, then the minimum number of queries is $log_2 n$. Note that this minimum number of queries can only be achieved when all minimal diagnoses are considered at once, which is intractable even for relatively small values of n.

However, in case probabilities of diagnoses are known we can reduce the number of queries by utilizing two effects:

Query	$\mathbf{D^P}$	$\mathbf{D^N}$	$\mathbf{D^\emptyset}$
$Q_1 : \{B(w)\}$	$\{\mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_1\}$	$\emptyset$
$Q_2 : \{B(w), C(w)\}$	$\{\mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_1, \mathcal{D}_2\}$	$\emptyset$
$Q_3 : \{B(w), C(w), Q(w)\}$	$\{\mathcal{D}_4\}$	$\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$	$\emptyset$

Table 15.2: Possible queries in Example 15.1

1. We can exploit diagnoses probabilities to assess the likelihood of each answer and the expected value of the information contained in the set of diagnoses after an answer is given.

2. Even if multiple diagnoses remain, further query generation may not be required if one diagnosis is highly probable and all other remaining diagnoses are highly improbable.

Example 15.2 Consider an ontology $\mathcal{O}$ with the terminology $\mathcal{T}$:

$$ax_1 : A_1 \sqsubseteq A_2 \sqcap M_1 \sqcap M_2 \qquad ax_4 : M_2 \sqsubseteq \forall s.A \sqcap D$$
$$ax_2 : A_2 \sqsubseteq \neg\exists s.M_3 \sqcap \exists s.M_2 \qquad ax_5 : M_3 \equiv B \sqcup C$$
$$ax_3 : M_1 \sqsubseteq \neg A \sqcap B$$

and the background theory containing the assertions $\mathcal{A} : \{A_1(w), A_1(u), s(u, w)\}$.

The ontology along with the background theory is inconsistent and the set of minimal conflict sets $CS = \{\langle ax_1, ax_3, ax_4 \rangle, \langle ax_1, ax_2, ax_3, ax_5 \rangle\}$. To restore consistency, the user should modify all axioms of at least one minimal diagnosis:

$$\mathcal{D}_1 : [ax_1] \qquad\qquad \mathcal{D}_3 : [ax_4, ax_5]$$
$$\mathcal{D}_2 : [ax_3] \qquad\qquad \mathcal{D}_4 : [ax_4, ax_2]$$

$\square$

Following the same approach as in Example 15.1, we compute a set of possible queries and corresponding partitions using the algorithm presented above. A set of possible irreducible queries for Example 15.2 and their partitions are presented in Table 15.3. These queries partition the set of diagnoses $\mathbf{D}$ in a way that makes the application of myopic strategies, such as "split-in-half", inefficient. A greedy algorithm based on such a heuristic would first select the first query Q_1, since there is no query that cuts the set of diagnoses in half. If $\mathcal{D}_4$ is the target diagnosis then Q_1 will be answered with *yes* by an oracle (see Figure 15.1). In the next iteration the algorithm would also choose a suboptimal query, the first untried query Q_2, since there is no partition that divides the diagnoses $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_4$ into two groups of equal size. Once again, the oracle answers *yes*, and the algorithm identifies query Q_4 to differentiate between $\mathcal{D}_1$ and $\mathcal{D}_4$.

Query	$\mathbf{D^P}$	$\mathbf{D^N}$	$\mathbf{D^\emptyset}$
$Q_1 : \{B \sqsubseteq M_3\}$	$\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_4\}$	$\{\mathcal{D}_3\}$	$\emptyset$
$Q_2 : \{B(w)\}$	$\{\mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_2\}$	$\{\mathcal{D}_1\}$
$Q_3 : \{M_1 \sqsubseteq B\}$	$\{\mathcal{D}_1, \mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_2\}$	$\emptyset$
$Q_4 : \{M_1(w), M_2(u)\}$	$\{\mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_1\}$	$\emptyset$
$Q_5 : \{A(w)\}$	$\{\mathcal{D}_2\}$	$\{\mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_1\}$
$Q_6 : \{M_2 \sqsubseteq D\}$	$\{\mathcal{D}_1, \mathcal{D}_2\}$	$\emptyset$	$\{\mathcal{D}_3, \mathcal{D}_4\}$
$Q_7 : \{M_3(u)\}$	$\{\mathcal{D}_4\}$	$\emptyset$	$\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$

Table 15.3: Possible queries in Example 15.2

However, in real-world settings the assumption that all axioms fail with the same probability is rarely the case. For example, Roussey et al. [RCVB09] present a list of "anti-patterns" where an anti-pattern is a set of axioms, such as $\{C1 \sqsubseteq \forall R.C2, C1 \sqsubseteq \forall R.C3, C2 \equiv \neg C3\}$ that corresponds to a minimal conflict set. The study performed by [RCVB09] shows that such conflict sets often occur in practice due to frequent misuse of certain language constructs like quantification or disjointness. Such studies are ideal sources for estimating prior fault probabilities. However, this is beyond the scope of our work presented in this part.

Our approach for computing the prior fault probabilities of axioms is inspired by [RDH$^+$04] and considers the syntax of a knowledge representation language, such as restrictions, conjunction, negation, etc. For instance, if a user frequently changes the universal to the existential quantifier and vice versa in order to restore coherency, then we can assume that axioms including such restrictions are more likely to fail than the other ones. In [RDH$^+$04] the authors report that in most cases inconsistent ontologies are created because users (a) mix up $\forall r.S$ and $\exists r.S$, (b) mix up $\neg \exists r.S$ and $\exists r.\neg S$, (c) mix up $\sqcup$ and $\sqcap$, (d) wrongly assume that classes are disjoint by default or overuse disjointness, or (e) wrongly apply negation. Observing that misuses of quantifiers are more likely than other failure patterns one might find that the axioms ax_2 and ax_4 are more likely to be faulty than ax_3 (because of the use of quantifiers), whereas ax_3 is more likely to be faulty than ax_5 and ax_1 (because of the use of negation).

Detailed justifications of diagnoses probabilities are given in the next section. However, let us assume some probability distribution of the faults according to the observations presented above such that: (a) the diagnosis $\mathcal{D}_2$ is the most probable one, i.e. single fault diagnosis of an axiom containing a negation; (b) although $\mathcal{D}_4$ is a double fault diagnosis, it follows $\mathcal{D}_2$ closely as its axioms contain quantifiers; (c) $\mathcal{D}_1$ and $\mathcal{D}_3$ are significantly less probable than $\mathcal{D}_4$ because conjunction/disjunction in ax_1 and ax_5 have a significantly lower fault probability than negation in ax_3. Taking this information into account asking query Q_1 is essentially useless because it is highly probable that the target diagnosis is either $\mathcal{D}_2$ or $\mathcal{D}_4$ and, therefore, it is highly probable that the oracle will respond with yes. Instead, asking Q_3 is more informative because regardless of the answer we can exclude one of the highly probable diagnoses, i.e. either $\mathcal{D}_2$ or $\mathcal{D}_4$. If the oracle responds to Q_3 with no then $\mathcal{D}_2$ is the only remaining diagnosis. However, if the oracle responds with yes, diagnoses $\mathcal{D}_4$, $\mathcal{D}_3$, and $\mathcal{D}_1$ remain, where $\mathcal{D}_4$ is significantly more probable compared to diagnoses $\mathcal{D}_3$ and $\mathcal{D}_1$. If the difference between the probabilities of the diagnoses is high enough such that $\mathcal{D}_4$ can be accepted as the target diagnosis, no additional questions are required. Obviously this strategy can lead to a substantial reduction in the number of queries compared to myopic approaches as we demonstrate in our evaluation.

Note that in real-world application scenarios failure patterns and their probabilities can be discovered by analyzing the debugging actions of a user in an ontology editor, like Protégé. Learning of fault probabilities can be used to "personalize" the query selection algorithm to prefer user-specific faults.
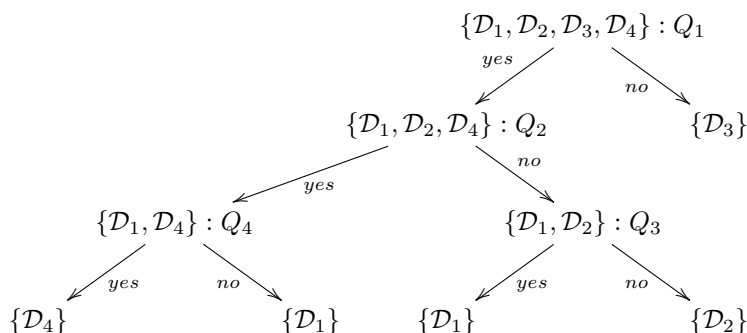

Figure 15.1: The search tree of the greedy algorithm

However, as our evaluation shows, even a rough estimate of the probabilities is capable of outperforming the "split-in-half" heuristic.

Chapter 16

Entropy-Based Query Selection

To select the best query we exploit a-priori failure probabilities of each axiom derived from the syntax of description logics or some other knowledge representation language, such as OWL. That is, the user is able to specify own beliefs in terms of the probability of syntax element such as $\forall$, $\exists$, $\sqcap$, etc. being erroneous; alternatively, the debugger can compute these probabilities by analyzing the frequency of various syntax elements in the target diagnoses of different debugging sessions. If no failure information is available then the debugger can initialize all of the probabilities with some small value. Compared to statistically well-founded probabilities, the latter approach provides a suboptimal but useful diagnosis discrimination process, as discussed in the evaluation.

Given the failure probabilities of all syntax elements $se \in \mathbf{S}$ of a knowledge representation language used in $\mathcal{O}$, we can compute the failure probability of an axiom $ax_i \in \mathcal{O}$

$$p(ax_i) = p(F_{se_1} \cup F_{se_2} \cup \cdots \cup F_{se_n})$$

where $F_{se_1} \ldots F_{se_n}$ represent the events that the occurrence of a syntax element se_j in ax_i is faulty. E.g. for ax_2 of Example 15.2 $p(ax_2) = p(F_{\sqsubseteq} \cup F_{\neg} \cup F_{\exists} \cup F_{\sqcap} \cup F_{\exists})$. Assuming that each occurrence of a syntax element fails independently, i.e. an erroneous usage of a syntax element se_k makes it neither more nor less probable that an occurrence of syntax element se_j is faulty, the failure probability of an axiom is computed as:

$$p(ax_i) = 1 - \prod_{se \in \mathbf{S}} (1 - F_{se})^{c(se)} \tag{16.1}$$

where $c(se_j)$ returns number of occurrences of the syntax element se_j in an axiom ax_i. If among other failure probabilities the user states that $p(F_{\sqsubseteq}) = 0.001, p(F_{\neg}) = 0.01, p(F_{\exists}) = 0.05$ and $p(F_{\sqcap}) = 0.001$ then $p(ax_2) = p(F_{\sqsubseteq} \cup F_{\neg} \cup F_{\exists} \cup F_{\sqcap} \cup F_{\exists}) = 0.108$.

Given the failure probabilities $p(ax_i)$ of axioms, the diagnosis algorithm first calculates the a-priori probability $p(\mathcal{D}_j)$ that $\mathcal{D}_j$ is the target diagnosis. Since all axioms fail independently, this probability can be computed as [dKW87]:

$$p(\mathcal{D}_j) = \prod_{ax_n \in \mathcal{D}_j} p(ax_n) \prod_{ax_m \in \mathcal{O} \backslash \mathcal{D}_j} 1 - p(ax_m) \tag{16.2}$$

The prior probabilities for diagnoses are then used to initialize an iterative algorithm that includes two main steps: (a) the selection of the best query and (b) updating the diagnoses probabilities given query feedback.

According to information theory the best query is the one that, given the answer of an oracle, minimizes the expected entropy of the set of diagnoses [dKW87]. Let $p(Q_i = yes)$ be the probability that query Q_i is answered with yes and $p(Q_i = no)$ be the probability for the answer no. Furthermore, let $p(\mathcal{D}_j|Q_i = yes)$ be the probability of diagnosis $\mathcal{D}_j$ after the oracle answers yes and $p(\mathcal{D}_j|Q_i = no)$ be the probability after the oracle answers no. The expected entropy after querying Q_i is:

$$H_e(Q_i) = \sum_{v \in \{yes, no\}} p(Q_i = v) \sum_{\mathcal{D}_j \in \mathbf{D}} -p(\mathcal{D}_j|Q_i = v) \log_2 p(\mathcal{D}_j|Q_i = v)$$

Based on a one-step-look-ahead information theoretic measure, the query which minimizes the expected entropy is considered best. This formula can be simplified to the following score function [dKW87] which we use to evaluate all available queries and select the one with the minimum score to maximize information gain:

$$sc(Q_i) = \sum_{v \in \{yes, no\}} \left[p(Q_i = v) \log_2 p(Q_i = v) \right] + p(\mathbf{D_i^\emptyset}) + 1 \tag{16.3}$$

where $v \in \{yes, no\}$ is a feedback of an oracle and $\mathbf{D_i^\emptyset}$ is the set of diagnoses which do not make any predictions for the query Q_i. The probability of the set of diagnoses $p(\mathbf{D_i^\emptyset})$ as well as of any other set of diagnoses $\mathbf{D_i}$ like $\mathbf{D_i^P}$ and $\mathbf{D_i^N}$ is computed as:

$$p(\mathbf{D_i}) = \sum_{\mathcal{D}_j \in \mathbf{D_i}} p(\mathcal{D}_j)$$

because by Definition 28.2, each diagnosis uniquely partitions all of the axioms of an ontology $\mathcal{O}$ into two sets, correct and faulty, and thus all diagnoses are mutually exclusive events.

Since, for a query Q_i, the set of diagnoses $\mathbf{D}$ can be partitioned into the sets $\mathbf{D_i^P}$, $\mathbf{D_i^N}$ and $\mathbf{D_i^\emptyset}$, the probability that an oracle will answer a query Q_i with either yes or no can be computed as:

$$\begin{aligned} p(Q_i = yes) &= p(\mathbf{D_i^P}) + p(\mathbf{D_i^\emptyset})/2 \\ p(Q_i = no) &= p(\mathbf{D_i^N}) + p(\mathbf{D_i^\emptyset})/2 \end{aligned} \tag{16.4}$$

Clearly this assumes that for each diagnosis of $\mathbf{D_i^\emptyset}$ *both outcomes are equally likely* and thus the probability that the set of diagnoses $\mathbf{D_i^\emptyset}$ predicts either $Q_i = yes$ or $Q_i = no$ is $p(\mathbf{D_i^\emptyset})/2$.

Following feedback v for a query Q_s, i.e. $Q_s = v$, the probabilities of the diagnoses must be updated to take the new information into account. The update is made using Bayes' rule for each $\mathcal{D}_j \in \mathbf{D}$:

$$p(\mathcal{D}_j|Q_s = v) = \frac{p(Q_s = v|\mathcal{D}_j)p(\mathcal{D}_j)}{p(Q_s = v)} \tag{16.5}$$

where the denominator $p(Q_s = v)$ is known from the query selection step (Equation 16.4) and $p(\mathcal{D}_j)$ is either a prior probability (Equation 16.2) or is a probability calculated using Equation 16.5 after a previous iteration of the debugging algorithm. We assign $p(Q_s = v|\mathcal{D}_j)$ as follows:

$$p(Q_s = v|\mathcal{D}_j) = \begin{cases} 1, & \text{if } \mathcal{D}_j \text{ predicted } Q_s = v; \\ 0, & \text{if } \mathcal{D}_j \text{ is rejected by } Q_s = v; \\ \frac{1}{2}, & \text{if } \mathcal{D}_j \in \mathbf{D_s^\emptyset} \end{cases}$$

Example 16.1   (Example 15.1 continued) Suppose that the debugger is not provided with any information about possible failures and therefore assumes that all syntax elements fail with the same probability

Query	Initial score	$Q_2 = yes$
$Q_1 : \{B(w)\}$	0.1887	**0**
$Q_2 : \{C(w)\}$	**0**	1
$Q_3 : \{Q(w)\}$	0.1887	1

Table 16.1: Expected scores for minimized queries ($p(ax_i) = 0.01$)

Query	Initial score
$Q_1 : \{B(w)\}$	**0.250**
$Q_2 : \{C(w)\}$	0.408
$Q_3 : \{Q(w)\}$	0.629

Table 16.2: Expected scores for minimized queries ($p(ax_1) = 0.025$, $p(ax_2) = p(ax_3) = p(ax_4) = 0.01$)

0.01 and therefore $p(ax_i) = 0.01$ for all $ax_i \in \mathcal{O}$. Using Equation 16.2 we can calculate probabilities for each diagnosis. For instance, $\mathcal{D}_1$ suggests that only one axiom ax_1 should be modified by the user. Hence, we can calculate the probability of diagnosis D_1 as $p(\mathcal{D}_1) = p(ax_1)(1-p(ax_2))(1-p(ax_3))(1-p(ax_4)) = 0.0097$. All other minimal diagnoses have the same probability, since every other minimal diagnosis suggests the modification of one axiom. To simplify the discussion we only consider minimal diagnoses for query selection. Therefore, the prior probabilities of the diagnoses can be normalized to $p(\mathcal{D}_j) = p(\mathcal{D}_j)/\sum_{\mathcal{D}_j \in \mathbf{D}} p(\mathcal{D}_j)$ and are equal to 0.25.

Given the prior probabilities of the diagnoses and a set of queries (see Table 15.2) we evaluate the score function (Equation 16.3) for each query. E.g. for the first query $Q_1 : \{B(w)\}$ the probability $p(\mathbf{D}^\emptyset) = 0$ and the probabilities of both the positive and negative outcomes are: $p(Q_1 = 1) = p(\mathcal{D}_2) + p(\mathcal{D}_3) + p(\mathcal{D}_4) = 0.75$ and $p(Q_1 = 0) = p(\mathcal{D}_1) = 0.25$. Therefore the query score is $sc(Q_1) = 0.1887$.

The scores computed during the initial stage (see Table 16.1) suggest that Q_2 is the best query. Taking into account that $\mathcal{D}_1$ is the target diagnosis the oracle answers *no* to the query. The additional information obtained from the answer is then used to update the probabilities of diagnoses using the Equation 16.5. Since $\mathcal{D}_1$ and $\mathcal{D}_2$ predicted this answer, their probabilities are updated, $p(\mathcal{D}_1) = p(\mathcal{D}_2) = 1/p(Q_2 = 1) = 0.5$. The probabilities of diagnoses $\mathcal{D}_3$ and $\mathcal{D}_4$ which are rejected by the oracle's answer are also updated, $p(\mathcal{D}_3) = p(\mathcal{D}_4) = 0$.

In the next iteration the algorithm recomputes the scores using the updated probabilities. The results show that Q_1 is the best query. The other two queries Q_2 and Q_3 are irrelevant since no information will be gained if they are asked. Given the oracle's negative feedback to Q_1, we update the probabilities $p(\mathcal{D}_1) = 1$ and $p(\mathcal{D}_2) = 0$. In this case the target diagnosis $\mathcal{D}_1$ was identified using the same number of steps as the "split-in-half" heuristic.

However, if the user specifies that the first axiom is more likely to fail, e.g. $p(ax_1) = 0.025$, then $Q_1 : \{B(w)\}$ will be selected first (see Table 16.2). The recalculation of the probabilities given the negative outcome $Q_1 = 0$ sets $p(\mathcal{D}_1) = 1$ and $p(\mathcal{D}_2) = p(\mathcal{D}_3) = p(\mathcal{D}_4) = 0$. Therefore the debugger identifies the target diagnosis in only one step. □

Example 16.2 (Example 15.2 continued) Suppose that in ax_4 the user specified $\forall s.A$ instead of $\exists s.A$ and $\neg \exists s.M_3$ instead of $\exists s.\neg M_3$ in ax_2. Therefore $\mathcal{D}_4$ is the target diagnosis. Moreover, assume that the debugger is provided with observations of three types of faults: (1) conjunction/disjunction occurs with probability $p_1 = 0.001$, (2) negation $p_2 = 0.01$, and (3) restrictions $p_3 = 0.05$. Using Equation 16.1 we can calculate the probability of the axioms containing an error: $p(ax_1) = 0.0019$, $p(ax_2) = 0.1074$, $p(ax_3) = 0.012$, $p(ax_4) = 0.051$, and $p(ax_5) = 0.001$. These probabilities are exploited to calculate the prior probabilities of the diagnoses (see Table 16.3) and to initialize the query selection process. To

Answers		$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$
Prior		0.0970	0.5874	0.0026	0.3130
$Q_3 = yes$		0.2352	0	0.0063	0.7585
$Q_3 = yes, Q_4 = yes$		0	0	0.0082	0.9918
$Q_3 = yes, Q_4 = yes, Q_1 = yes$		0	0	0	1

Table 16.3: Probabilities of diagnoses after answers

Queries	Initial	$Q_3 = yes$	$Q_3 = yes, Q_4 = yes$
$Q_1 : \{B \sqsubseteq M_3\}$	0.974	0.945	**0.931**
$Q_2 : \{B(w)\}$	0.151	0.713	1
$Q_3 : \{M_1 \sqsubseteq B\}$	**0.022**	1	1
$Q_4 : \{M_1(w), M_2(u)\}$	0.540	**0.213**	1
$Q_5 : \{A(w)\}$	0.151	0.713	1
$Q_6 : \{M_2 \sqsubseteq D\}$	0.686	0.805	1
$Q_7 : \{M_3(u)\}$	0.759	0.710	0.970

Table 16.4: Expected scores for queries

simplify matters we focus on the set of minimal diagnoses.

In the first iteration the algorithm determines that Q_3 is the best query and asks the oracle whether $\mathcal{O}_t \models \{M_1 \sqsubseteq B\}$ is true or not (see Table 16.4). The obtained information is then used to recalculate the probabilities of the diagnoses and to compute the next best subsequent query, i.e. Q_4, and so on. The query process stops after the third query, since $\mathcal{D}_4$ is the only diagnosis that has the probability $p(\mathcal{D}_4) > 0$.

Given the feedback of the oracle $Q_4 = yes$ for the second query, the updated probabilities of the diagnoses show that the target diagnosis has a probability of $p(\mathcal{D}_4) = 0.9918$ whereas $p(\mathcal{D}_3)$ is only 0.0082. In order to reduce the number of queries a user can specify a threshold, e.g. $\sigma = 0.95$. If the absolute difference in probabilities of two most probable diagnoses is greater than this threshold, the query process stops and returns the most probable diagnosis. Therefore, in this example the debugger based on the entropy query selection requires less queries than the "split-in-half" heuristic. Note that already after the first answer $Q_3 = yes$ the most probable diagnosis $\mathcal{D}_4$ is three times more likely than the second most probable diagnosis $\mathcal{D}_1$. Given such a great difference we could suggest to stop the query process after the first answer if the user would set $\sigma = 0.65$. □

Chapter 17

Implementation Details

The iterative ontology debugger (Algorithm 11) takes a faulty ontology $\mathcal{O}$ as input. Optionally, a user can provide a set of axioms $\mathcal{B}$ that are known to be correct as well as a set P of axioms that must be entailed by the target ontology and a set N of axioms that must not. If these sets are not given, the corresponding input arguments are initialized with $\emptyset$. Moreover, the algorithm takes a set FP of fault probabilities for axioms $ax_i \in \mathcal{O}$, which can be computed as described in Chapter 16 by exploiting knowledge about typical user errors. Alternatively, if no estimates of such probabilities are available, all probability values can be initialized using a small constant. We show the results of such a strategy in our evaluation section. The two other arguments σ and n are used to improve the performance of the algorithm. σ specifies the diagnosis acceptance threshold, i.e. the minimum difference in probabilities between the most likely and second-most likely diagnoses. The parameter n defines the maximum number of most probable diagnoses that should be considered by the algorithm during each iteration. A further performance gain in Algorithm 11 can be achieved if we approximate the set of the n most probable diagnoses with the set of the n most probable *minimal* diagnoses, i.e. we neglect non-minimal diagnoses. We call this set of at most n most probable minimal diagnoses the *leading diagnoses*. Note, under the reasonable assumption that the fault probability of each axiom $p(ax_i)$ is less than 0.5, for every non-minimal diagnosis ND a minimal diagnosis $\mathcal{D} \subset ND$ exists which from Equation 16.2 is more probable than ND. Consequently the query selection algorithm presented here operates on the set of minimal diagnoses instead of all diagnoses (i.e. non-minimal diagnoses are excluded). However, the algorithm can be adapted with moderate effort to also consider non-minimal diagnoses.

We use the approach proposed by Friedrich et al. [FS05] to compute diagnoses and employ the combination of two algorithms, QUICKXPLAIN [Jun04] and HS-TREE [Rei87]. In a standard implementation the latter is a breadth-first search algorithm that takes an ontology $\mathcal{O}$, sets P and N, and the maximum number of most probable minimal diagnoses n as an input. The algorithm generates minimal hitting sets using minimal conflict sets, which are computed on-demand. This is motivated by the fact that in some circumstances a subset of all minimal conflict sets is sufficient for generating a subset of all required minimal diagnoses. For instance, in Example 15.2 the user wants to compute only $n = 2$ leading minimal diagnoses and a minimal conflict search algorithm returns CS_1. In this case HS-TREE identifies two required minimal diagnoses $\mathcal{D}_1$ and $\mathcal{D}_2$ and avoiding the computation of the minimal conflict set CS_2. Of course, in the worst case, when all minimal diagnoses have to be computed the algorithm should compute all minimal conflict sets. In addition, the HS-TREE generation reuses minimal conflict sets in order to avoid unnecessary computations. Thus, in the real-world scenarios we evaluated (see Table 18.1), less than 10 minimal conflict sets were contained in the faulty ontologies having at most 13 elements while the maximal cardinality of minimal diagnoses was observed to be at most 9. Therefore, space limitations were not a problem for the breadth-first generation. However, for scenarios involving diagnoses of greater

---

Algorithm 11 ONTODEBUGGING($\mathcal{O}, \mathcal{B}, P, N, FP, n, \sigma$)

Input: ontology $\mathcal{O}$, set $\mathcal{B}$ of background axioms, set P of sets of logical sentences to be entailed, set N of sets of logical sentences not to be entailed, set FP of fault probabilities for axioms, maximum number n of most probable minimal diagnoses, acceptance threshold σ

Output: a diagnosis $\mathcal{D}$

1: $DP \leftarrow \emptyset$
2: $QH \leftarrow \emptyset$
3: $T \leftarrow \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$
4: **while** BELOWTHRESHOLD(DP, σ) $\wedge$ GETSCORE(T) $\neq 1$ **do**
5: $\quad DP \leftarrow$ HS-TREE($\mathcal{O}, \mathcal{B}, P, N, FP, QH, n$)
6: $\quad T \leftarrow$ SELECTQUERY($DP, \mathcal{O}, \mathcal{B}, P$)
7: $\quad Q \leftarrow$ GETQUERY(T)
8: $\quad$ **if** $Q = \emptyset$ **then**
9: $\quad\quad$ **exit loop**
10: $\quad$ **if** GETANSWER($\mathcal{O}_t \models Q$) **then**
11: $\quad\quad P \leftarrow P \cup \{Q\}$
12: $\quad$ **else**
13: $\quad\quad N \leftarrow N \cup \{Q\}$
14: $\quad QH \leftarrow QH \cup \{T\}$
15: **return** MOSTPROBABLEDIAGNOSIS(DP)

---

cardinalities iterative-deepening strategies could be applied.

In our implementation of HS-TREE we use the uniform-cost search strategy. Given additional information in terms of axiom fault probabilities FP, the algorithm expands a leaf node in a search-tree if it is an element of the path corresponding to the maximum probability hitting set of minimal conflict sets computed so far. The probability of each minimal hitting set can be computed using Equation 16.2. Consequently, the algorithm computes a set of diagnoses ordered by their probability starting from the most probable one. HS-TREE terminates if either the n most probable minimal diagnoses are identified or no further minimal diagnoses can be found. Thus the algorithm computes at most n minimal diagnoses regardless of the number of all minimal diagnoses.

HS-TREE uses QUICKXPLAIN to compute required minimal conflicts. This algorithm, given a set of axioms AX and a set of correct axioms $\mathcal{B}$ returns a minimal conflict set $CS \subseteq AX$, or $\emptyset$ if axioms $AX \cup \mathcal{B}$ are consistent. In the worst case, to compute a minimal conflict QUICKXPLAIN performs $2k(\log(s/k) + 1)$ consistency checks, where k is the size of the generated minimal conflict set and s is the number of axioms in the ontology. In the best case only $\log(s/k) + 2k$ are performed [Jun04]. Importantly, the size of the ontology is contained in the $\log$ function. Therefore, the time needed for consistency checks in our test ontologies remained below 0.2 seconds, even for real world knowledge bases with thousands of axioms. The maximum time to compute a minimal conflict was observed in the Sweet-JPL ontology and took approx. 5 seconds (see Table 18.2).

In order to take past answers into account the HS-TREE updates the prior probabilities of the diagnoses by evaluating Equation 16.5. All required data is stored in the query history QH as well as in the sets P and N. When complete, HS-TREE returns a set of tuples of the form $\langle \mathcal{D}_i, p(\mathcal{D}_i) \rangle$ where $\mathcal{D}_i$ is contained in the set of the n most probable minimal diagnoses (leading diagnoses) and $p(\mathcal{D}_i)$ is its probability calculated using Equation 16.2 and Equation 16.5.

In the query-selection phase Algorithm 11 calls SELECTQUERY function (Algorithm 12) to generate a tuple $T = \langle Q, \mathbf{D^P}, \mathbf{D^N}, \mathbf{D^\emptyset} \rangle$, where Q is the minimum score query (Equation 16.3) and $\mathbf{D^P}, \mathbf{D^N}$ and $\mathbf{D^\emptyset}$ the sets of diagnoses constituting the partition. The generation algorithm carries out a depth-first search, removing the top element of the set D and calling itself recursively to generate all possible

---

Algorithm 12 SELECTQUERY$(DP, \mathcal{O}, \mathcal{B}, P)$

Input: set DP of tuples $\langle \mathcal{D}_i, p(\mathcal{D}_i) \rangle$, ontology $\mathcal{O}$, set of background axioms $\mathcal{B}$, set P of sets of logical sentences
    that must be entailed by the target ontology
Output: a tuple $\left\langle Q, \mathbf{D^P}, \mathbf{D^N}, \mathbf{D^\emptyset} \right\rangle$

 1: $\mathbf{D} \leftarrow$ GETDIAGNOSES(DP)
 2: $T \leftarrow$ GENERATE$(\emptyset, \mathbf{D}, \mathcal{O}, \mathcal{B}, P, DP)$
 3: **return** MINIMIZEQUERY(T)

 4: **procedure** GENERATE$(\mathbf{D^P}, D, \mathcal{O}, \mathcal{B}, P, DP)$ **returns** a tuple $\left\langle Q, \mathbf{D^P}, \mathbf{D^N}, \mathbf{D^\emptyset} \right\rangle$
 5:      **if** $D = \emptyset$ **then**
 6:          $\mathbf{D} \leftarrow$ GETDIAGNOSES(DP)
 7:          **return** CREATEQUERY$(\mathbf{D^P}, \mathcal{O}, \mathcal{B}, P, \mathbf{D})$
 8:      $\mathcal{D} \leftarrow$ POP(D)
 9:      $left \leftarrow$ GENERATE$(\mathbf{D^P}, D, \mathcal{O}, \mathcal{B}, P, DP)$
10:      $right \leftarrow$ GENERATE$(\mathbf{D^P} \cup \{\mathcal{D}\}, D, \mathcal{O}, \mathcal{B}, P, DP)$
11:      **if** GETSCORE$(left, DP) <$ GETSCORE$(right, DP)$ **then**
12:          **return** $left$
13:      **else**
14:          **return** $right$

---

subsets of the leading diagnoses. The set of leading diagnoses $\mathbf{D}$ is extracted from the set of tuples DP by the GETDIAGNOSES function. In each leaf node of the search tree the GENERATE function calls CREATEQUERY creates a query given a set of diagnoses $\mathbf{D^P}$ by computing common entailments and partitioning the set of diagnoses $\mathbf{D} \setminus \mathbf{D^P}$, as described in Section 15. If a query for the set $\mathbf{D^P}$ does not exist (i.e. there are no common entailments) or $\mathbf{D^P} = \emptyset$ then CREATEQUERY returns an empty tuple $T = \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$. In all inner nodes of the tree the algorithm selects a tuple that corresponds to a query with the minimum score as found using the GETSCORE function. This function may implement the entropy-based measure (Equation 16.3), "split-in-half" or any other preference criteria. Given an empty tuple $T = \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$ the function returns the highest possible score of a used measure. In general, CREATEQUERY is called 2^n times, where we set $n = 9$ in our evaluation. Furthermore, for each leading diagnosis not in $\mathbf{D^P}$, CREATEQUERY has to check if the associated query is entailed. If a query is not entailed, a consistency check has to be performed. Entailments are determined by classification/realization and a subset check of the generated sentences. Common entailments are computed by exploiting the intersection of entailments for each diagnosis contained in $\mathbf{D^P}$. Note that the entailments for each leading diagnosis are computed just once and reused in for subsequent calls of CREATEQUERY.

In the function MINIMIZEQUERY, the query Q of the resulting tuple $\langle Q, \mathbf{D^P}, \mathbf{D^N}, \mathbf{D^\emptyset} \rangle$ is iteratively reduced by applying QUICKXPLAIN such that sets $\mathbf{D^P}$, $\mathbf{D^N}$ and $\mathbf{D^\emptyset}$ are preserved. This is implemented by replacing the consistency checks performed by QUICKXPLAIN with checks that ensure that the reduction of the query preserves the partition. In order to check if a partition is preserved, a consistency/entailment check is performed for each element in $\mathbf{D^N}$ and $\mathbf{D^\emptyset}$. Elements of $\mathbf{D^P}$ need not be checked because these elements entail the query and therefore any reduction. In the worst case $n(2k \log(s/k) + 2k)$ consistency checks have to be performed in MINIMIZEQUERY where k is the length of the minimized query. Entailments of leading diagnoses are reused.

Algorithm 11 invokes the function GETQUERY to obtain the query from the tuple stored in T and calls GETANSWER to query the oracle. Depending on the answer, Algorithm 11 extends either the set P or the set N and thus excludes diagnoses not compliant with the query answer from the results of HS-TREE in further iterations. Note, the algorithm can be easily adapted to allow the oracle to reject a query if the

answer is unknown. In this case the algorithm proceeds with the next best query (w.r.t. the GETSCORE function) until no further queries are available.

Algorithm 11 stops if the difference in the probabilities of the top two diagnoses is greater than the acceptance threshold σ or if no query can be used to differentiate between the remaining diagnoses (i.e. the score of the minimum score query equals to the maximum score of the used measure). The most probable diagnosis is then returned to the user. If it is impossible to differentiate between a number of highly probable minimal diagnoses, the algorithm returns a set that includes all of them. Moreover, in the first case (termination due to σ), the algorithm can continue if the user is not satisfied with the returned diagnosis and at least one further query exists.

Additional performance improvements can be achieved by using greedy strategies in Algorithm 12. The idea is to guide the search such that a leaf node of the left-most branch of a search tree contains a set of diagnoses $\mathbf{D}^{\mathbf{P}}$ that might result in a tuple $\langle Q, \mathbf{D}^{\mathbf{P}}, \mathbf{D}^{\mathbf{N}}, \mathbf{D}^{\emptyset} \rangle$ with a low-score query. This method is based on the property of Equation 16.3 that $sc(Q) = 0$ if

$$\sum_{\mathcal{D}_i \in \mathbf{D}^{\mathbf{P}}} p(\mathcal{D}_i) = \sum_{\mathcal{D}_j \in \mathbf{D}^{\mathbf{N}}} p(\mathcal{D}_j) = 0.5 \quad \text{and} \quad p(\mathbf{D}^{\emptyset}) = 0$$

Consequently, the query selection problem can be presented as a two-way number partitioning problem: given a set of numbers, divide them into two sets such that the difference between the sums of the numbers in each set is as small as possible. The Complete Karmarkar-Karp (CKK) algorithm [Kor98], which is one of the best algorithms developed for the two-way partitioning problem, corresponds to an extension of the Algorithm 12 with a set differencing heuristic [KKLO86]. The algorithm stops if the optimal solution to the two-way partitioning problem is found or if there are no further subsets to be investigated. In the latter case the best found solution is returned.

The main drawback of applying CKK to the query selection process is that none of the pruning techniques can be used. Also even if the algorithm finds an optimal solution to the two-way partitioning problem there just might be no query for a found set of diagnoses $\mathbf{D}^{\mathbf{P}}$. Moreover, since the algorithm is complete it still has to investigate all subsets of the set of diagnoses in order to find the minimum score query. To avoid this exhaustive search we extended CKK with an additional termination criterion: the search stops if a query is found with a score below some predefined threshold γ. In our evaluation section we demonstrate substantial savings by applying the CKK partitioning algorithm.

To sum up, the proposed method depends on the efficiency of the classification/realization system and consistency/coherency checks given a particular ontology. The number of calls to a reasoning system can be reduced by decreasing the number of leading diagnoses n. However, the more leading diagnoses provide the more data for generating the next best query. Consequently, by varying the number of leading diagnoses it is possible to balance runtime with the number of queries needed to isolate the target diagnosis.[32]

---

[32]The source code as well as precompiled binaries can be downloaded from http://rmbd.googlecode.com. The package also includes a Protégé-plugin implementing the methods as described.

Chapter 18

Evaluation

We evaluated our approach using the real-world ontologies presented in Table 18.1 with the aim of demonstrating its applicability real-world settings. In addition, we employed generated examples to perform controlled experiments where the number of minimal diagnoses and their cardinality could be varied to make the identification of the target diagnosis more difficult. Finally, we carried out a set of tests using randomly modified large real-world ontologies to provide some insights on the scalability of the suggested debugging method.

For the first test we created a generator which takes a consistent and coherent ontology, a set of fault patterns together with their probabilities, the minimum number of minimum cardinality diagnoses m, and the required cardinality $|\mathcal{D}_t|$ of these minimum cardinality diagnoses as inputs. We also assumed that the target diagnosis has cardinality $|\mathcal{D}_t|$. The output of the generator is an alteration of the input ontology for which at least the given number of minimum cardinality diagnoses with the required cardinality exist. Furthermore, to introduce inconsistencies (incoherencies), the generator applies fault patterns randomly to the input ontology depending on their probabilities.

In this experiment we took five fault patterns from a case study reported by Rector et al. [RDH$^+$04] and assigned fault probabilities according to their observations of typical user errors. Thus we assumed that in cases (a) and (b) (see Section 15), where an axiom includes some roles (i.e. property assertions), axiom descriptions are faulty with a probability of 0.025, in cases (c) and (d) 0.01 and in case (e) 0.001. In each iteration, the generator randomly selected an axiom to be altered and applied a fault pattern. Following this, another axiom was selected using the concept taxonomy and altered correspondingly to introduce an inconsistency (incoherency). The fault patterns were randomly selected in each step using

	Ontology	DL	Axioms	#C/#P/#I	#CS/min/max	#D/min/max	Domain
1.	Chemical	$\mathcal{ALCHF}^{(D)}$	144	48/20/0	6/5/6	6/1/3	Chemical elements
2.	Koala	$\mathcal{ALCON}^{(D)}$	44	21/5/6	3/4/4	10/1/3	Training
3.	Sweet-JPL	$\mathcal{ALCHOF}^{(D)}$	2579	1537/121/50	1/13/13	13/1/1	Earthscience
4.	miniTambis	$\mathcal{ALCN}$	173	183/44/0	3/2/6	48/3/3	Biological science
5.	University	$\mathcal{SOIN}^{(D)}$	49	30/12/4	4/3/5	90/3/4	Training
6.	Economy	$\mathcal{ALCH}^{(D)}$	1781	339/53/482	8/3/4	864/4/8	Mid-level
7.	Transportation	$\mathcal{ALCH}^{(D)}$	1300	445/93/183	9/2/6	1782/6/9	Mid-level

Table 18.1: Diagnosis results for several of the real-world ontologies presented in [KPHS07]. #C/#P/#I are the number of concepts, properties and individuals in each ontology. #CS/min/max are the number of conflict sets, and their minimum and maximum cardinality. The same notation is used for diagnoses #D/min/max. The ontologies are available upon request.

Ontology		Leading diagnoses			All diagnoses		
		Consistency	Conflicts	Diagnoses	Consistency	Conflicts	Diagnoses
Chemical	time	0/3/8	90/107/128	1/97/326	0/3/18	105/130/179	2/126/402
	calls	264	6	7	262	6	7
			runtime: 723			runtime: 892	
Koala	time	0/1/3	19/25/30	0/11/70	0/2/4	24/30/37	0/12/105
	calls	74	3	10	75	3	11
			runtime: 120			runtime: 148	
Sweet-JPL	time	1/31/112	5185/5185/5185	0/586/5332	31/106/195	5192/5192/5192	1/438/5319
	calls	187	1	10	195	1	14
			runtime: 5991			runtime: 6312	
miniTambis	time	0/5/14	84/157/210	0/57/504	1/5/15	88/167/225	3/19/537
	calls	111	3	10	189	3	49
			runtime: 586			runtime: 1027	
University	time	0/2/3	31/41/54	0/20/157	0/2/5	37/46/60	2/5/200
	calls	126	4	10	283	4	91
			runtime: 205			runtime: 536	
Economy	time	1/12/26	410/460/569	0/282/2085	1/9/80	418/510/681	16/25/1929
	calls	239	6	10	2064	8	865
			runtime: 2857			runtime: 25369	
Transportaton	time	0/11/58	237/438/683	0/352/3176	1/9/130	222/429/636	16/29/6394
	calls	337	7	10	3966	9	1783
			runtime: 3671			runtime: 65010	

Table 18.2: Min/avg/max time and calls required to compute the nine leading most probable diagnoses as well as all diagnoses for the real-world ontologies. Values are given for each stage, i.e. consistency checking, computation of minimal conflicts and minimal diagnoses, together with the total runtime needed to compute the diagnoses. All time values are 15 trial averages and are given in milliseconds.

the probabilities provided above.

For instance, given the description of a randomly selected concept A and the fault pattern "misuse of negation", we added the construct $\sqcap \neg X$ to the description of A, where X is a new concept name. Next, we randomly selected concepts B and S such that $S \sqsubseteq A$ and $S \sqsubseteq B$ and added $\sqcap X$ to the description of B. During the generation process, we applied the HS-TREE algorithm after each introduction of an incoherency/inconsistency to control two parameters: the minimum number of minimal cardinality diagnoses in the ontology and their cardinality. The generator continues to introduce incoherences/inconsistencies until the specified parameter values are reached. For instance, if the minimum number of minimum cardinality diagnoses is equal to $m = 6$ and their cardinality is $|\mathcal{D}_t| = 4$, then the generated ontology will include at least 6 diagnoses of cardinality 4 and possibly some additional number of minimal diagnoses of higher cardinalities.

The resulting faulty ontology as well as the fault patterns and their probabilities were inputs for the ontology debugger. The acceptance threshold σ was set to 0.95 and the number of most probable minimal diagnoses n was set to 9. In addition, one of the minimal diagnoses with the required cardinality was randomly selected as the target diagnosis. Note, the target ontology is not equal to the original ontology, but rather a corrected version of the altered one in which the faulty axioms were repaired by replacing them with their original (correct) versions according to the target diagnosis. The tests were performed
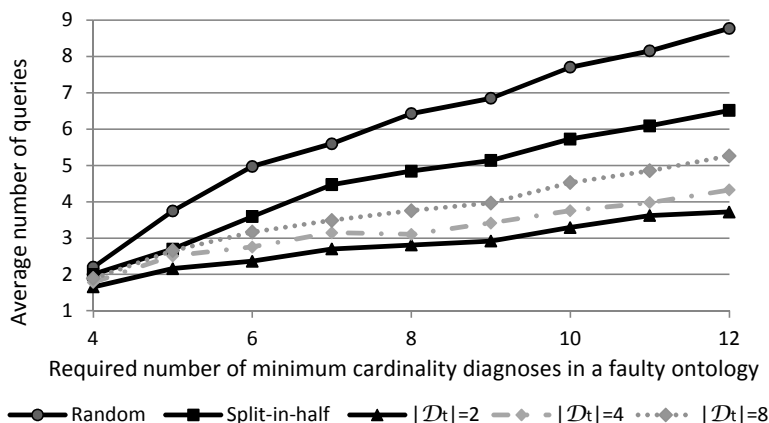
Figure 18.1: Average number of queries required to select the target diagnosis $\mathcal{D}_t$ with threshold $\sigma = 0.95$. Random and "split-in-half" are shown for the cardinality of minimal diagnoses $|\mathcal{D}_t| = 2$.

using the ontologies bike2 to bike9, bcs3, galen and galen2 from Racer's benchmark suite[33].

The average results of the evaluation performed on each test ontology (presented in Figure 18.1) show that the entropy-based approach outperforms the "split-in-half" heuristic as well as the random query selection strategy by more than 50% for the $|\mathcal{D}_t| = 2$ case due to its ability to estimate the probabilities of diagnoses and to stop once the target diagnosis crossed the acceptance threshold. On average the algorithm required 8 seconds to generate a query. In addition, Figure 18.1 shows that the number of queries required increases as the cardinality of the target diagnosis increases, regardless of the method. Despite this, the entropy-based approach remains better than the "split-in-half" method for diagnoses with increasing cardinality. The approach did however require more queries to discriminate between high cardinality diagnoses because in such cases more minimal conflicts were generated. Consequently, the debugger should consider more minimal diagnoses in order to identify the target one.

For the next test we selected seven real-world ontologies described in Tables 18.1 and 18.2[34]. Performance of both the entropy-based and "split-in-half" selection strategies was evaluated using a variety of different prior fault probabilities to investigate under which conditions the entropy-based method should be preferred.

In our experiments we distinguished between three different distributions of prior fault probabilities: extreme, moderate and uniform (see Figure 18.2 for an example). The *extreme distribution* simulates a situation in which very high failure probabilities are assigned to a small number of syntax elements. That is, the provider of the estimates is quite sure that exactly these elements are causing a fault. For instance, it may be well known that a user has problems formulating restrictions in OWL whereas all other elements, such as subsumption and conjunction, are well understood. In the case of a *moderate distribution* the estimates provide a slight bias towards some syntax elements. This distribution has the same motivation as the extreme one, however, in this case the probability estimator is less sure about the sources of possible errors in axioms. Both extreme and moderate distributions correspond to the exponential distribution with $\lambda = 1.75$ and $\lambda = 0.5$ respectively. The *uniform distribution* models the situation where no prior fault probabilities are provided and the system assigns equal probabilities to all syntax elements found in a faulty ontology. Of course the prior probabilities of diagnoses may not reflect the actual situation. Therefore, for each of the three distributions we differentiate between good, average and bad cases. In the *good case* the estimates of the prior fault probabilities are correct and the

---

[33] Available at http://www.racer-systems.com/products/download/benchmark.phtml

[34] All experiments were performed on a PC with Core2 Duo (E8400), 3 Ghz with 8 Gb RAM, running Windows 7 and Java 6.
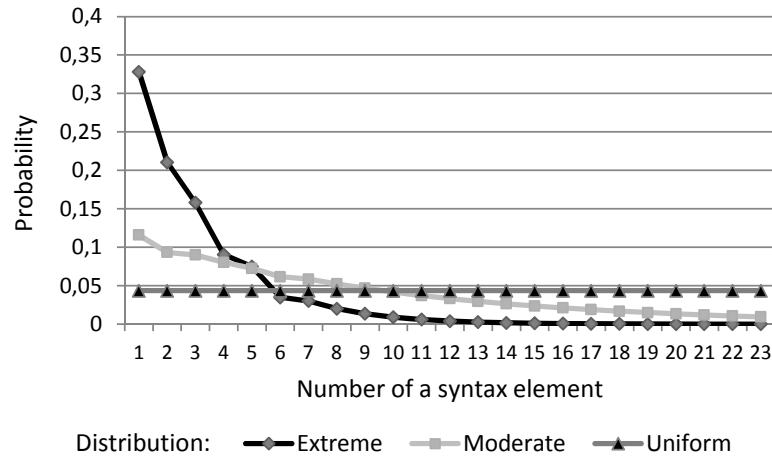
Figure 18.2: Example of prior fault probabilities of syntax elements sampled from extreme, moderate and uniform distributions.

target diagnosis is assigned a high probability. The *average case* corresponds to the situation when the target diagnosis is neither favored nor penalized by the priors. In the *bad case* the prior distribution is unreasonable and disfavors the target diagnosis by assigning it a low probability.

We executed 30 tests for each of the combinations of the distributions and cases with an acceptance threshold $\sigma = 0.85$ and a required number of most probable minimal diagnoses $n = 9$. Each iteration started with the generation of a set of prior fault probabilities of syntax elements by sampling from a selected distribution (extreme, moderate or uniform). Given the priors we computed the set of all minimal diagnoses $\mathbf{D}$ of a given ontology and selected the target one according to the chosen case (good, average or bad). In the good case the prior probabilities favor the target diagnosis and, therefore, it should be selected from the diagnoses with high probability. The set of diagnoses was ordered according to their probabilities and the algorithm iterated through the set starting from the most probable element. In the first iteration the most probable minimal diagnosis $\mathcal{D}_1$ is added to the set G. In next iteration j a diagnosis $\mathcal{D}_j$ was added to the set G if $\sum_{i \leq j} p(\mathcal{D}_i) \leq \frac{1}{3}$ and to the set A if $\sum_{i \leq j} p(\mathcal{D}_i) \leq \frac{2}{3}$. The obtained set G contained all most probable diagnoses which we considered as good. All diagnoses in the set $A \setminus G$ were classified as average and the remaining diagnoses $\mathbf{D} \setminus A$ as bad. Depending on the selected case we randomly selected one of the diagnoses as the target from the appropriate set.

The results of the evaluation presented in Table 18.3 show that the entropy-based query selection approach clearly outperforms "split-in-half" in good and average cases for the three probability distributions. The average time required by the debugger to perform such basic operations as consistency checking, computation of minimal conflicts and diagnoses is presented in Table 18.4. The results indicate that on average at most 17 seconds required to compute up to 9 minimal diagnoses and a query. Moreover, the number of axioms in a query remains reasonable in most of the cases stays bounds, i.e. between 1 and 4 axioms per query.

In the uniform case better results were observed since the diagnoses have different cardinality and structure, i.e. they include different syntax elements. Consequently, even if equal probabilities for all syntax elements (uniform distribution) are given, the probabilities of diagnoses are different. Axioms with a greater number of syntax elements receive a higher fault probability. Also, diagnoses with a smaller cardinality in many cases receive a higher probability. This information provides enough bias to favor the entropy-based method.

In the bad case, where the target diagnosis received a low probability and no information regarding the

Entropy-based query selection

Ontology	Case	Extreme			Moderate			Uniform		
		min	avg	max	min	avg	max	min	avg	max
	Good	1	**1.63**	3	1	**1.7**	2	1	**1.83**	2
Chemical	Avg.	1	**1.87**	4	1	**1.73**	3	1	**1.7**	2
	Bad	2	3.03	4	2	3.03	4	2	3.17	4
	Good	1	**1.7**	3	1	**2.4**	4	1	**2.67**	3
Koala	Avg.	1	**1.8**	3	1	**2.37**	4	1	**2.4**	3
	Bad	1	3.5	6	2	4.33	7	3	4.13	5
	Good	1	**3.27**	7	2	**3.43**	7	3	**3.87**	7
Sweet-JPL	Avg.	1	**3.5**	6	1	4.03	7	3	4.07	6
	Bad	3	3.93	6	2	4.03	6	3	**3.37**	4
	Good	1	**2.37**	4	2	**2.73**	4	2	**2.77**	3
miniTambis	Avg.	1	**2.53**	4	2	**4.03**	8	3	**4.53**	7
	Bad	3	6.43	11	3	7.93	17	5	9.03	13
	Good	1	**2.7**	4	3	**3.83**	7	3	**4.4**	8
University	Avg.	1	**3.4**	6	3	7.03	12	4	**7.27**	10
	Bad	5	9.13	15	5	9.7	14	6	10.03	14
	Good	1	**3.2**	11	3	**3.1**	4	3	**3.93**	6
Economy	Avg.	1	**4.63**	14	3	**5.57**	12	5	**6.5**	8
	Bad	8	12.3	19	6	11.5	21	7	11.67	19
	Good	1	**5.63**	14	1	**6.97**	12	3	**9.5**	14
Transportation	Avg.	1	**6.9**	16	1	**7.73**	12	3	**8.73**	14
	Bad	3	**12.4**	18	8	**12.8**	20	3	**12.1**	18

"Split-in-half" query selection

Ontology	Case	min	avg	max	min	avg	max	min	avg	max
	Good	2	2.63	3	2	2.7	3	2	2.53	3
Chemical	Avg.	2	2.63	3	2	2.67	3	2	2.77	3
	Bad	2	**2.63**	3	2	**2.6**	3	2	**2.4**	3
	Good	3	3.3	4	3	3.3	4	3	3.47	4
Koala	Avg.	3	3.33	4	3	3.2	4	3	3.23	4
	Bad	3	**3.43**	4	3	**3.4**	4	3	**3.5**	4
	Good	3	3.83	4	3	3.8	4	4	4	4
Sweet-JPL	Avg.	3	3.57	4	3	**3.8**	4	3	**3.47**	4
	Bad	3	**3.87**	4	3	**3.8**	4	3	3.8	4
	Good	4	5.33	6	4	5	6	4	4	4
miniTambis	Avg.	4	5.1	6	4	4.93	7	5	5.43	7
	Bad	5	**5.93**	8	4	**5.8**	7	5	**6.3**	7
	Good	4	5.93	8	4	6	8	4	5.43	8
University	Avg.	4	5.87	7	5	**6.73**	9	6	7.37	8
	Bad	5	**6.97**	9	5	**7.2**	9	5	**7**	8
	Good	6	7.87	11	6	7.4	10	6	7.5	10
Economy	Avg.	6	8	12	5	7.63	12	6	8.73	13
	Bad	9	**11.50**	14	6	**11.1**	14	8	**11.3**	15
	Good	5	8.03	13	5	7.3	11	6	11.43	18
Transportation	Avg.	3	9	16	5	9.4	13	5	11.43	18
	Bad	10	12.67	19	7	13	19	6	13.8	20

Table 18.3: Minimum, average and maximum number of queries required by the entropy-based and "split-in-half" query selection methods to identify the target diagnosis in real-world ontologies. Ontologies are ordered by the number of diagnoses.

Ontology	Good			Average			Bad		
	DT	QT	QL	DT	QT	QL	DT	QT	QL
Chemical	459.33	117.67	3	461.33	121	3.34	256.67	75.67	2.19
Koala	88.33	1308.33	3.47	92	1568.67	3.90	56.33	869.33	2.36
Sweet-JPL	2387.33	691.67	1.48	2272	926	1.61	2103	1240.33	1.57
miniTabmis	481.33	2764.33	3.27	398.33	2892	2.53	238.67	3223	1.76
University	189.33	822.67	3.91	145	903.33	2.82	113	872	2.11
Economy	2953.33	6927	3.06	3239	8789	3.80	3083	8424.67	1.58
Transportation	6577.33	9426.33	2.37	7080.67	10135.33	2.29	7186.67	9599.67	1.64

Table 18.4: Average time required to compute at most nine minimal diagnoses (DT) and a query (QT) in each iteration, as well as the average number of axioms in a query after minimization (QL). The averages are shown for extreme, moderate and uniform distributions using the entropy-based query selection method. Time is measured in milliseconds.
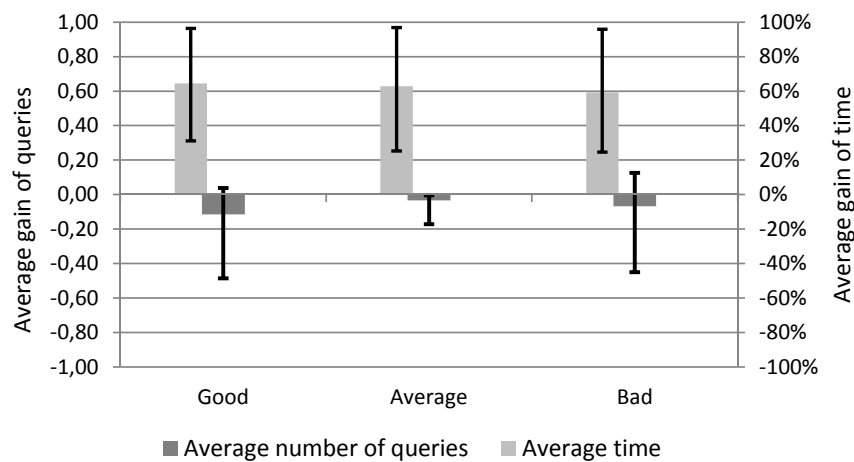

Figure 18.3: Average time/query gain resulting from the application of the extended CKK partitioning algorithm. The whiskers indicate the maximum and minimum possible average gain of queries/time using extended CKK.

prior fault probabilities was given, we observed that the performance of the entropy-method improved as more queries were posed. In particular, in the University ontology the performance is essentially similar (7.27 vs. 7.37) whereas in the Economy and Transportation ontology the entropy-based method can save and average of two queries.

"Split-in-half" appears to be particularly inefficient in all good, average and bad cases when applied to ontologies with a large number of minimal diagnoses, such as Economy and Transportation. The main problem is that no stop criteria can be used with the greedy method as it is unable to provide any ordering on the set of diagnoses. Instead, the method continues until no further queries can be generated, i.e. only one minimal diagnosis exists or there are no discriminating queries. Conversely, the entropy-based method is able to improve its probability estimates using Bayes-updates as more queries are answered and to exploit the differences in the probabilities in order to decide when to stop.

The most significant gains are achieved for ontologies with many minimal diagnoses and for the average and good cases, e.g. the target diagnosis is within the first or second third of the minimal diagnoses ranked by their prior probability. In these cases the entropy-based method can save up to 60% of the queries.

Ontology	Cton	Opengalen-no-propchains
Axioms	33203	9664
DL	$\mathcal{SHF}$	$\mathcal{ALEHIF}^{(D)}$
#CS/min/max	6/3/7	9/5/8
#D/min/max	15/1/5	110/2/6
Consistency	5/209/1078	1/98/471
QuickXplain	17565/20312/38594	7634/10175/12622
Diagnosis	1/5285/38594	10/1043/19543
Overall runtime	146186	119973

Table 18.5: Statistics for the real-world ontologies used in the stress-tests measured for a single random alteration. #CS/min/max are the number of minimal conflict sets, and their minimum and maximum cardinality. The same notation is used for diagnoses #D/min/max. The minimum/average/maximum time required to make a consistency check (Consistency), compute a minimal conflict set (QuickXplain) and a minimal diagnosis are measured in milliseconds. Overall runtime indicates the time required to compute all minimal diagnoses in milliseconds.

Ontology	#Query	Overall	QT	DT	QL
Good					
Cton	3	176828	6918	52237	4
Opengalen-no-propchains	8	154145	2349	22905	4
Average					
Cton	4	177383	6583	52586	3
Opengalen-no-propchains	7	151048	3752	21344	4
Bad					
Cton	5	190407	5742	35608	1
Opengalen-no-propchains	14	177728	1991	11319	3

Table 18.6: Average values measured for extreme, moderate and uniform distributions in each of the good, average and bad cases. #Query is the number of queries required to find the target diagnosis. Overall runtime as well as the time required to compute a query (QT) and at least nine minimal diagnoses (DT) are given in milliseconds. Query length (QL) shows the average number of axioms in a query.

Therefore, we can conclude that even rough estimates of the prior fault probabilities are sufficient, provided that the target diagnosis is not significantly penalized. Even if no fault probabilities are available and there are many minimal diagnoses, the entropy-based method is advantageous. The differences between probabilities of individual syntax elements appears not to influence the results of the query selection process and affect only the number of outliers, i.e. cases in which the diagnosis approach required either few or many queries compared to the average.

Another interesting observation is that often both methods eliminated more than n diagnoses in one iteration. For instance, in the case of the Transportation ontology both methods were able to remove hundreds of minimal diagnoses with a small number of queries. This behavior appears to stem from relations between the diagnoses. That is, the addition of a query to either P or N allows the method to remove not only the diagnoses in sets $\mathbf{D}^\mathbf{P}$ or $\mathbf{D}^\mathbf{N}$, but also some unobserved diagnoses that were not in any of the sets of n leading diagnoses computed by HS-TREE. Given the sets P and N, HS-TREE automatically invalidates all diagnoses which do not fulfill the requirements (see Definition 28.2).
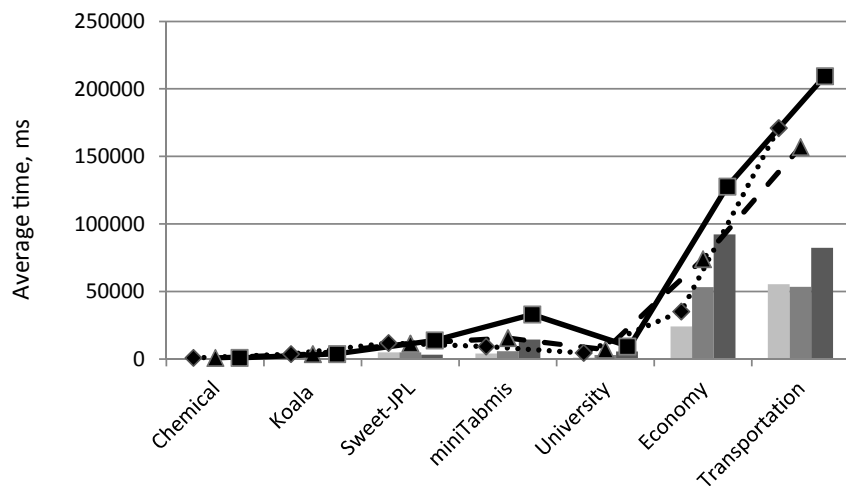
The extended CKK method presented in Chapter 17 was evaluated in the same settings as the complete Algorithm 12 with acceptance threshold $\gamma = 0.1$. The obtained results presented in Figure 18.3 show that the extended CKK method decreases the length of a debugging session by at least 60% while requiring
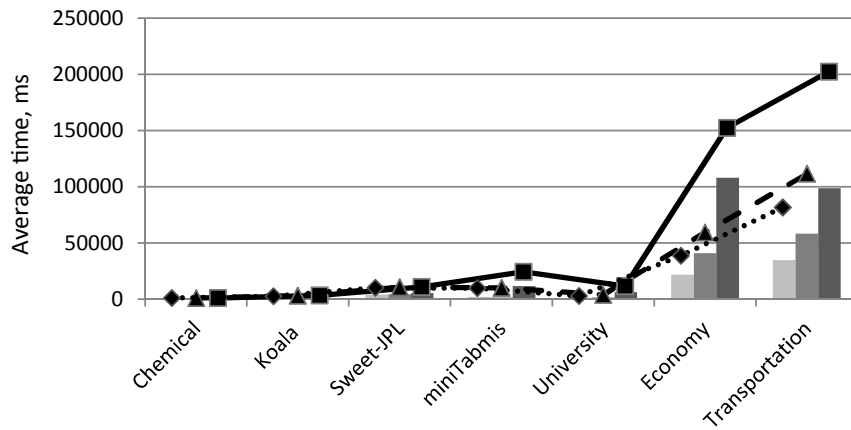
a) Extreme distribution
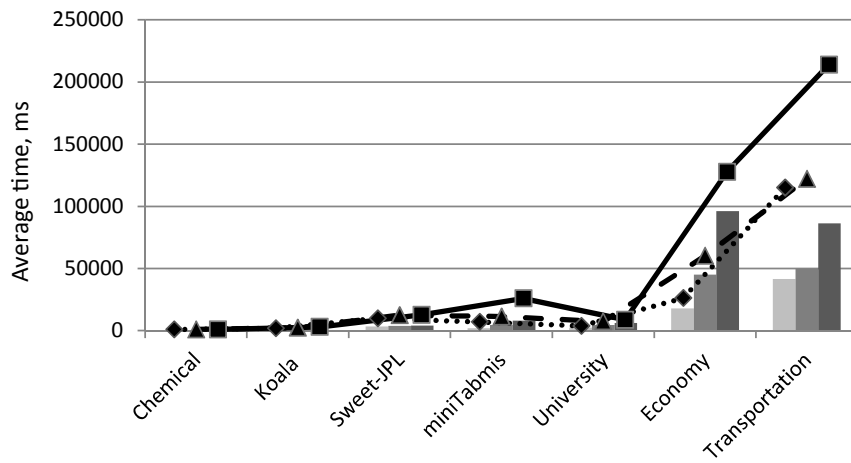
b) Moderate distribution

c) Uniform distribution

| CKK | Good | Average | Bad |
| SelectQuery | Good | Average | Bad |

Figure 18.4: Average time required to identify the target diagnosis using CKK and brute force query selection algorithms.

on average 0.1 queries more than Algorithm 12. In some cases (mostly for the uniform distribution) the debugger using CKK search required even fewer queries than Algorithm 12 because of the inherent uncertainty of the domain. The plot of the average time required by Algorithm 12 and CKK to identify the target diagnosis presented in Figure 18.4 shows that the application of the latter can reduce runtime significantly.

In the last experiment we tried to simulate an expert developing large real-world ontologies[35] as described in Table 18.5. Often in such settings an expert makes small changes to the ontology and then runs the reasoner to verify that the changes are valid, i.e. the ontology is consistent and its entailments are correct. To simulate this scenario we used the generator described in the first experiment to introduce 1 to 3 random changes that would make the ontology incoherent. Then, for each modified ontology, we performed 15 tests using the fault distributions as in the second test. The results obtained by the entropy-based query selection method using CKK for query computation are presented in Table 18.6. These results show that the method can be used for analysis of large ontologies with over 33000 axioms while requiring a user to wait for only a minute to compute the next query.

---

[35]The ontologies taken from TONES repository http://owl.cs.manchester.ac.uk/repository

Chapter 19

Related Work

Despite the range of ontology diagnosis methods available (see [SHCH07, KPHS07, FS05]), to the best of our knowledge no interactive ontology debugging methods, such as our "split-in-half" or entropy-based methods, have been proposed so far. The idea of ranking of diagnoses and proposing a target diagnosis is presented in [KPSCG06]. This method uses a number of measures such as: (a) the frequency with which an axiom appears in conflict sets, (b) impact on an ontology in terms of its "lost" entailments when an axiom is modified or removed, (c) ranking of test cases, (d) provenance information about axioms, and (e) syntactic relevance. For each axiom in a conflict set, these measures are evaluated and combined to produce a rank value. These ranks are then used by a modified HS-TREE algorithm to identify diagnoses with a minimal rank. However, the method fails when a target diagnosis cannot be determined reliably with the given a-priori knowledge. In our work required information is acquired until the target diagnosis can be identified with confidence. In general, the work of [KPSCG06] can be combined with the ideas presented in our work as axiom ranks can be taken into account together with other observations for calculating the prior probabilities of the diagnoses.

The idea of selecting the next best query based on the expected entropy was exploited in the generation of decisions trees in [Qui86] and further refined for selecting measurements in the model-based diagnosis of circuits in [dKW87]. We extend these methods to query selection in the domain of ontology debugging.

In the area of debugging logic programs, Shapiro [Sha83] developed debugging methods based on query answering. Roughly speaking, Shapiro's method aims to detect one fault at a time by querying an oracle about the intended behavior of a Prolog program at hand. In our terminology, for each answer that must not be entailed this diagnosis approach generates one conflict at a time by exploiting the proof tree of a Prolog program. The method then identifies a query that splits the conflict in half. Our approach can deal with multiple diagnoses and conflicts simultaneously which can be exploited by query generation strategies such as "split-in-half" and entropy-based methods. Whereas the "split-in-half" strategy splits the set of diagnoses in half, Shapiros's method focuses on one conflict. Furthermore, the exploitation of failure probabilities is not considered in [Sha83]. However, Shapiro's method includes the learning of new clauses in order to cover not entailed answers. Interleaving discrimination of diagnoses and learning of descriptions is currently not considered in our approach because of their additional computational costs.

From a general point of view Shapiro's method can be seen as a prominent example of inductive logic programming (ILP) including systems such as [MB88, Mug95]. In particular, [Mug95] proposes inverse entailments combined with general to specific search through a refinement graph with the goal of generating a theory (hypothesis) which covers the examples and fulfills additional properties. Compared to ILP, the focus of our work lies on the theory revision. However, our knowledge representation languages are variants of description logics and not logic programs. Moreover, our method aims to discover axioms

which must be changed while minimizing user interaction. Preferences of theory changes are expressed by probabilities which are updated through Bayes' rule. Other preferences based on plausible extensions of the theory were not considered, again because of their computational costs.

Although model-based diagnosis has also been applied to logic programs [CFD93], constraint knowledge bases [FFJS04] and hardware descriptions [FSW99], none of these approaches propose a query generation method to discriminate between diagnoses.

Chapter 20

Summary and Conclusions

In this part we presented an approach to the interactive debugging of ontologies. This approach is applicable to any knowledge representation language with monotonic semantics. We showed that the axioms generated by classification and realization reasoning services can be exploited to generate queries which differentiate between diagnoses. For selecting the best next query we proposed two strategies: The "split-in-half" strategy prefers queries which allow eliminating a half of the leading diagnoses. The entropy-based strategy employs information theoretic concepts to exploit knowledge about the likelihood of axioms to be faulty. Based on the probability of an axiom containing an error we predict the (expected) information gain produced by a query result, enabling us to select the best subsequent query according to a one-step-lookahead entropy-based scoring function. We described the implementation of an interactive debugging algorithm and compared the entropy-based method with the "split-in-half" strategy. Our experiments showed a significant reduction in the number of queries required to identify the target diagnosis when the entropy-based method is applied. Depending on the quality of the given prior fault probabilities the required number of queries could be reduced by up to 60%.

In order to evaluate the robustness of the entropy-based method we experimented with different prior fault probability distributions as well as different qualities of the prior probabilities. Furthermore, we investigated cases where knowledge about failure probabilities is missing or inaccurate. In case such knowledge is unavailable, the entropy-based methods ranks the diagnoses based on the number of syntax elements contained in an axiom and the number of axioms in a diagnosis. Given that this is a reasonable guess (i.e. the target diagnosis is not at the lower end of the diagnoses ranked by their prior probabilities), the entropy-based method outperformed "split-in-half". Moreover, even if the initial guess is not reasonable, the entropy-based method improves the accuracy of the probabilities as more questions are asked. Furthermore, the applicability of the approach to real-world ontologies containing thousands of axioms was demonstrated by an extensive set of evaluations which are publicly available.

Part V

Minimizing User Interaction in Ontology Debugging

A reinforcement learning query selection strategy (RIO) that makes the presented debugging system robust against the usage of low-quality fault information is presented and thoroughly analyzed in this part which is based on the publications [RSFF13, RSFF12, RSFF11, SRF11] published in *Web Reasoning and Rule Systems (RR-2013)*, in the *Proceedings of the 7th International Workshop on Ontology Matching (OM-2012)*, in the *Proceedings of the Joint Workshop on Knowledge Evolution and Ontology Dynamics 2011 (EvoDyn2011)* and in *DX 2011 - 22nd International Workshop on Principles of Diagnosis*, respectively.

Chapter 21

Introduction to the Problem

The foundation for widespread adoption of Semantic Web technologies is a broad community of ontology developers which is not restricted to experienced knowledge engineers. Instead, domain experts from diverse fields should be able to create ontologies incorporating their knowledge as autonomously as possible. The resulting ontologies are required to fulfill some minimal quality criteria, usually consistency, coherency and no undesired entailments, in order to grant successful deployment. However, the correct formulation of logical descriptions in ontologies is an error-prone task which accounts for a need for assistance in ontology development in terms of ontology debugging tools. Usually, such tools [SHCH07, KPHS07, FS05, HPS08] use model-based diagnosis [Rei87] to identify sets of faulty axioms, called diagnoses, that need to be modified or deleted in order to meet the imposed quality requirements. The major challenge inherent in the debugging task is often a substantial number of alternative diagnoses.

In [SFFR12] this issue is tackled by letting the user take action during the debugging session by answering queries about entailments and non-entailments of the desired ontology. These answers pose constraints to the validity of diagnoses and thus help to sort out incompliant diagnoses step-by-step. In addition, a Bayesian approach is used to continuously readjust the fault probabilities by means of the additional information given by the user. The user effort in this interactive debugging procedure is strongly affected by the quality of the initially provided meta information, i.e. prior knowledge about fault probabilities of a user w.r.t. particular logical operators. To get this under control, the selection of queries shown to the user can be varied correspondingly. To this end, two essential paradigms for choosing the next "best" query have been proposed, split-in-half and entropy-based.

In order to opt for the optimal strategy, however, the quality of the meta information, i.e. good or bad (which means high or low probability of the correct solution), must be known in advance. This would, however, implicate the pre-knowledge of the initially unknown solution. Entropy-based methods can make optimal profit from exploiting properly adjusted initial fault probabilities (high potential), whereas they can completely fail in the case of weak prior information (high risk). The split-in-half technique, on the other hand, manifests constant behavior independently of the probabilities given (no risk), but lacks the ability to leverage appropriate fault information (no potential). This matter of fact is witnessed by the evaluation we conducted, which shows that an unsuitable combination of meta information and query selection strategy can result in a substantial increase of more than 2000% w.r.t. number of queries to a user. So, there is a need to either (1) guarantee a sufficiently suited choice of prior fault information, or (2) to manage the "risk" of unsuitable method selection. The task of (1) might not be a severe problem in a debugging scenario involving a faulty ontology developed by a single expert, since the meta information might be extracted from the logs of previous sessions, if available, or specified by the expert based on their experience w.r.t. own faults. However, realization of task (1) is a major issue in scenarios involving

automatized systems producing (parts of) ontologies, e.g. ontology alignment and ontology learning, or numerous users collaborating in modeling an ontology, where the choice of reasonable meta information is rather unclear. Therefore, we focus on accomplishing task (2).

The contribution of this part is a new *RI*sk *O*ptimization reinforcement learning method (RIO), which allows to minimize user interaction throughout a debugging session on average compared to existing strategies, for any quality of meta information (high potential at low risk). By virtue of its learning capability, our approach is optimally suited for debugging ontologies where only vague or no meta information is available. A learning parameter is constantly adapted based on the information gathered so far. On the one hand, our method takes advantage of the given meta information as long as good performance is achieved. On the other hand, it gradually gets more independent of meta information if suboptimal behavior is measured.

Experiments on two datasets of faulty real-world ontologies show the feasibility, efficiency and scalability of RIO. The evaluation will indicate that, on average, RIO is the best choice of strategy for both good and bad meta information with savings as to user interaction of up to 80%.

The problem specification, basic concepts and a motivating example are provided in Chapter 22. Chapter 23 explains the suggested approach and gives implementation details. Evaluation results are described in Chapter 24. Related work is discussed in Chapter 25. Chapter 26 concludes.

Chapter 22

Motivation and Basic Concepts

First we provide an informal introduction to ontology debugging, particularly addressing readers unfamiliar with the topic. Later we introduce precise formalizations. We assume the reader to be familiar with description logics [BCM$^+$07].

Ontology debugging deals with the following problem: Given is an ontology $\mathcal{O}$ which does not meet postulated requirements R, e.g. $R = \{\text{coherency}, \text{consistency}\}$. $\mathcal{O}$ is a set of axioms formulated in some monotonic knowledge representation language, e.g. OWL DL. The task is to find a subset of axioms in $\mathcal{O}$, called diagnosis, that needs to be altered or eliminated from the ontology in order to meet the given requirements. The presented approach to ontology debugging does not rely upon a specific knowledge representation formalism, it solely presumes that it is logic-based and monotonic. Additionally, the existence of sound and complete procedures for deciding logical consistency and for calculating logical entailments is assumed. These procedures are used as a black box. For OWL DL, e.g., both functionalities are provided by a standard DL-reasoner.

A diagnosis is a hypothesis about the state of each axiom in $\mathcal{O}$ of being either correct or faulty. Generally, there are many diagnoses for one and the same faulty ontology $\mathcal{O}$. The problem is then to figure out the single diagnosis, called target diagnosis $\mathcal{D}^*$, that complies with the knowledge to be modeled by the intended ontology. In interactive ontology debugging we assume a user, e.g. the author of the faulty ontology or a domain expert, interacting with an ontology debugging system by answering queries about entailments of the desired ontology, called the target ontology $\mathcal{O}^*$. The target ontology can be understood as $\mathcal{O}$ minus the axioms of $\mathcal{D}^*$ plus a set of axioms needed to preserve the desired entailments, called positive test cases. Note that the user is not expected to know $\mathcal{O}^*$ explicitly (in which case there would be no need to consult an ontology debugger), but implicitly in that they are able to answer queries about $\mathcal{O}^*$.

A query is a set of axioms and the user is asked whether the conjunction of these axioms is entailed by $\mathcal{O}^*$. Every positively (negatively) answered query constitutes a positive (negative) test case fulfilled by $\mathcal{O}^*$. The set of positive (entailed) and negative (non-entailed) test cases is denoted by P and N, respectively. So, P and N are sets of sets of axioms, which can be, but do not need to be, initially empty. Test cases can be seen as constraints $\mathcal{O}^*$ must satisfy and are therefore used to gradually reduce the search space for valid diagnoses. Roughly, the overall procedure consists of (1) computing a predefined number of diagnoses, (2) gathering additional information by querying the user, (3) incorporating this information to prune the search space for diagnoses, and so forth, until a stopping criterion is fulfilled, e.g. one diagnosis $\mathcal{D}^*$ has overwhelming probability.

The general debugging setting we consider also envisions the opportunity for the user to specify some background knowledge $\mathcal{B}$, i.e. a set of axioms that are known to be correct. $\mathcal{B}$ is then incorporated in the calculations throughout the ontology debugging procedure, but no axiom in $\mathcal{B}$ may take part in a

diagnosis. For example, in case the user knows that a subset of axioms in $\mathcal{O}$ is definitely sound, all axioms in this subset are added to $\mathcal{B}$ before initiating the debugging session. The advantage of this over simply not considering the axioms in $\mathcal{B}$ at all is, that the semantics of axioms in $\mathcal{B}$ is not lost and can be exploited, e.g., in query generation. $\mathcal{B}$ and $\mathcal{O} \setminus \mathcal{B}$ partition the original ontology into a set of correct and possibly incorrect axioms, respectively. In the debugging session, only $\mathcal{O} := \mathcal{O} \setminus \mathcal{B}$ is used to search for diagnoses. This can reduce the search space for diagnoses substantially. Another application of background knowledge could be the reuse of an existing ontology to support successful debugging. For example, when formulating an ontology about medical terms, a thoroughly curated reference ontology $\mathcal{B}$ could be leveraged to find own formulations contradicting the correct ones in $\mathcal{B}$, which would not be found without integration of $\mathcal{B}$ into the debugging procedure.

More formally, ontology debugging can be defined in terms of a diagnosis problem instance, for which we search for solutions, i.e. diagnoses, that enable to formulate the target ontology:

Definition 22.1 (Diagnosis Problem Instance, Target Ontology). *Let $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ be an ontology with terminological axioms $\mathcal{T}$ and assertional axioms $\mathcal{A}$, $\mathcal{B}$ a set of axioms which are assumed to be correct (background knowledge), R a set of requirements to $\mathcal{O}$, P and N respectively a set of positive and negative test cases, where each test case $p \in P$ and $n \in N$ is a set of axioms. Then we call the tuple $\langle \mathcal{O}, \mathcal{B}, P, N \rangle_R$ a* diagnosis problem instance (DPI). *An ontology $\mathcal{O}^*$ is called* target ontology *w.r.t. $\langle \mathcal{O}, \mathcal{B}, P, N \rangle_R$ iff all the following conditions hold:*

$$\begin{aligned} \forall\, r \in R &\quad : \quad \mathcal{O}^* \cup \mathcal{B} \text{ fulfills } r \\ \forall\, p \in P &\quad : \quad \mathcal{O}^* \cup \mathcal{B} \models p \\ \forall\, n \in N &\quad : \quad \mathcal{O}^* \cup \mathcal{B} \not\models n. \end{aligned}$$

Definition 22.2 (Diagnosis). *We call $\mathcal{D} \subseteq \mathcal{O}$ a* diagnosis *w.r.t. a DPI $\langle \mathcal{O}, \mathcal{B}, P, N \rangle_R$ iff $(\mathcal{O} \setminus \mathcal{D}) \cup (\bigcup_{p \in P} p)$ is a target ontology w.r.t. $\langle \mathcal{O}, \mathcal{B}, P, N \rangle_R$. A diagnosis $\mathcal{D}$ w.r.t. a DPI is* minimal *iff there is no $\mathcal{D}' \subset \mathcal{D}$ such that $\mathcal{D}'$ is a diagnosis w.r.t. this DPI. The set of minimal diagnoses w.r.t. a DPI is denoted by* $\mathbf{mD}$.

Note that a diagnosis $\mathcal{D}$ gives complete information about the correctness of each axiom $ax_k \in \mathcal{O}$, i.e. all $ax_i \in \mathcal{D}$ are assumed to be faulty and all $ax_j \in \mathcal{O} \setminus \mathcal{D}$ are assumed to be correct.

Example 22.1   Consider $\mathcal{O} := \mathcal{T} \cup \mathcal{A}$ with terminological axioms $\mathcal{T} := \mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}_{12}$:

$\mathcal{O}_1$	$ax_1 :$	$PhD \sqsubseteq Researcher$
	$ax_2 :$	$Researcher \sqsubseteq DeptEmployee$
$\mathcal{O}_2$	$ax_3 :$	$PhDStudent \sqsubseteq Student$
	$ax_4 :$	$Student \sqsubseteq \neg DeptMember$
$\mathcal{M}_{12}$	$ax_5 :$	$PhDStudent \sqsubseteq PhD$
	$ax_6 :$	$DeptEmployee \sqsubseteq DeptMember$

and an assertional axiom $\mathcal{A} = \{PhDStudent(s)\}$, where $\mathcal{M}_{12}$ is an automatically generated set of axioms serving as semantic links between $\mathcal{O}_1$ and $\mathcal{O}_2$. The given ontology $\mathcal{O}$ is inconsistent since it describes s as both a $DeptMember$ and not.

Let us assume that the assertion $PhDStudent(s)$ is considered as correct and is thus added to the background theory, i.e. $\mathcal{B} := \mathcal{A}$, and that no test cases are initially specified, i.e. the sets P and N are empty. For the resulting DPI $\langle \mathcal{T}, \mathcal{A}, \emptyset, \emptyset \rangle_{\{\text{coherence}\}}$ the set of minimal diagnoses $\mathbf{mD} = \{\mathcal{D}_1 : [ax_1], \mathcal{D}_2 : [ax_2], \mathcal{D}_3 : [ax_3], \mathcal{D}_4 : [ax_4], \mathcal{D}_5 : [ax_5], \mathcal{D}_6 : [ax_6]\}$. $\mathbf{mD}$ can be computed by a diagnosis algorithm such as the one presented in [FS05]. $\qquad\square$

With six minimal diagnoses for only six ontology axioms, this example already gives an idea that in many cases $|\mathbf{mD}|$ can get very large. Note that generally the computation of *all* minimal diagnoses w.r.t. a given DPI is not feasible within reasonable time due to the complexity of the underlying algorithms. Therefore, in practice, especially in an interactive scenario where reaction time is essential, a set of *leading diagnoses* $\mathbf{D} \subseteq \mathbf{mD}$ is considered as a representative for $\mathbf{mD}$.[36] Concerning the optimal number of leading diagnoses, a trade-off between representativeness and complexity of associated computations w.r.t. $\mathbf{D}$ needs to be found.

Without any prior knowledge in terms of diagnosis fault probabilities or specified test cases, each diagnosis in $\mathbf{D}$ is equally likely to be the target diagnosis $\mathcal{D}^*$. In other words, for each $\mathcal{D} \in \mathbf{D}$ w.r.t. the DPI $\langle \mathcal{T}, \mathcal{A}, \emptyset, \emptyset \rangle_{\{\text{coherence}\}}$, the ontology $(\mathcal{O} \setminus \mathcal{D}) \cup (\bigcup_{p \in P} p)$ meets all the conditions defining a target ontology. However, besides postulating coherence the user might want the target ontology to entail that s is a student as well as a researcher, i.e. $\mathcal{O}^* \models t_1$ where $t_1 := \{Researcher(s), Student(s)\}$. Formulating t_1 as a positive test case yields the DPI $\langle \mathcal{T}, \mathcal{A}, \{t_1\}, \emptyset \rangle_{\{\text{coherence}\}}$, for which only diagnoses $\mathcal{D}_2, \mathcal{D}_4, \mathcal{D}_6 \in \mathbf{D}$ are valid and enable to formulate a corresponding $\mathcal{O}^*$. All other diagnoses in $\mathbf{D}$ are ruled out by the fact that $t_1 \in P$, which means they have a probability of zero of being the target diagnosis. If $t_1 \in N$, in contrast, this would imply that $\mathcal{D}_2, \mathcal{D}_4, \mathcal{D}_6$ had to be rejected.

So, it depends on the test cases specified by a user which diagnosis will finally be identified as target diagnosis. Also, the order in which test cases are specified, is crucial. For instance, consider the test cases $t_1 := \{PhD(s)\}$ and $t_2 := \{Student(s)\}$. If $t_1 \in P$ is specified before $t_2 \in N$, then $t_1 \in P$ is redundant, since the only diagnosis agreeing with $t_2 \in N$ is $\mathcal{D}_3$ which preserves also the entailment t_1 in the resulting target ontology $\mathcal{O}^* = (\mathcal{O} \setminus \mathcal{D}_3) \cup \emptyset$ without explicating it as a positive test case.

Since it is by no means trivial to get the right – in the sense of most informative – test cases formulated in the proper order such that the number of test cases necessary to detect the target diagnosis is minimized, interactive debugging systems offer the functionality to automatize selection of test cases. The benefit is that the user can just concentrate on "answering" the provided test cases which means assigning them to either P or N. We call such automatically generated test cases queries. The theoretical foundation for the application of queries is the fact that $\mathcal{O} \setminus \mathcal{D}_i$ and $\mathcal{O} \setminus \mathcal{D}_j$ for $\mathcal{D}_i \neq \mathcal{D}_j \in \mathbf{D}$ entail different sets of axioms.

Definition 22.3 (Query, Partition). *Let $\mathbf{D}$ be a set of minimal diagnoses w.r.t. a DPI $\langle \mathcal{O}, \mathcal{B}, P, N \rangle_R$ and $\mathcal{O}_i^* := (\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup (\bigcup_{p \in P} p)$ for $\mathcal{D}_i \in \mathbf{D}$. Then a set of axioms $X_j \neq \emptyset$ is called a* query *w.r.t. $\mathbf{D}$ iff $\mathbf{D}_j^+ := \{\mathcal{D}_i \in \mathbf{D} \mid \mathcal{O}_i^* \models X_j\} \neq \emptyset$ and $\mathbf{D}_j^- := \{\mathcal{D}_i \in \mathbf{D} \mid \exists x \in N \cup R : \mathcal{O}_i^* \cup X_j \text{ violates } x\} \neq \emptyset$. The (unique) partition of a query X_j is denoted by $\langle \mathbf{D}_j^+, \mathbf{D}_j^-, \mathbf{D}_j^0 \rangle$ where $\mathbf{D}_j^0 = \mathbf{D} \setminus (\mathbf{D}_j^+ \cup \mathbf{D}_j^-)$. $\mathbf{X_D}$ terms a set of queries and associated partitions w.r.t. $\mathbf{D}$ in which one and the same partition of $\mathbf{D}$ occurs at most once and only if there is an associated query for this partition.*

Note that, in general, there can be n_q queries for a particular partition of $\mathbf{D}$ where n_q can be zero or some positive integer. We are interested in (1) only those partitions for each of which $n_q \geq 1$ and (2) only one query for each such partition. The set $\mathbf{X_D}$ includes elements such that (1) and (2) holds. $\mathbf{X_D}$ for a given set of minimal diagnoses $\mathbf{D}$ w.r.t. a DPI can be generated as shown in Algorithm 13. In each iteration, given a set of diagnoses $\mathbf{D}_k^+ \subset \mathbf{D}$, common entailments[37] $X_k := \{e \mid \forall \mathcal{D}_i \in \mathbf{D}_k^+ : \mathcal{O}_i^* \models e\}$ are computed (GETENTAILMENTS) and used to classify the remaining diagnoses in $\mathbf{D} \setminus \mathbf{D}_k^+$ to obtain the partition $\langle \mathbf{D}_k^+, \mathbf{D}_k^-, \mathbf{D}_k^0 \rangle$ associated with X_k. Then, if the partition $\langle \mathbf{D}_k^+, \mathbf{D}_k^-, \mathbf{D}_k^0 \rangle$ does not already occur in $\mathbf{X_D}$ (INCLUDESPARTITION), the query X_k is minimized [SFFR12] (MINIMIZEQUERY) such that its partition is preserved, yielding a query $X_k' \subseteq X_k$ such that any $X_k'' \subset X_k'$ is not a query or has

---

[36]So, we will speak of $\mathbf{D}$ instead of $\mathbf{mD}$ throughout this work. Note that the restriction to a subset of $\mathbf{mD}$ does not necessarily have implications on the completeness of the associated ontology debugging algorithm. E.g., the algorithm can be iterative and recompute new diagnoses on demand and nevertheless guarantee completeness (as the algorithm presented in this work).

[37]Note, when we speak of entailments throughout this work, we address (only) the *finite* set of entailments computed by the classification and realization services of a DL-reasoner.

Algorithm 13 Generation of Queries and Partitions

Input: DPI $\langle \mathcal{O}, \mathcal{B}, P, N \rangle_R$, set of minimal diagnoses $\mathbf{D}$ w.r.t. $\langle \mathcal{O}, \mathcal{B}, P, N \rangle_R$
Output: a set of queries and associated partitions $\mathbf{X_D}$

1: $\mathbf{X_D} \leftarrow \emptyset$
2: **for** $\mathbf{D}_k^+ \subset \mathbf{D}$ **do**
3:     $X_k \leftarrow \text{GETENTAILMENTS}(\mathcal{O}, \mathcal{B}, P, \mathbf{D}_k^+)$
4:     **if** $X_k \neq \emptyset$ **then**
5:         **for** $\mathcal{D}_r \in \mathbf{D} \setminus \mathbf{D}_k^+$ **do**
6:             **if** $\mathcal{O}_r^* \models X_k$ **then**
7:                 $\mathbf{D}_k^+ \leftarrow \mathbf{D}_k^+ \cup \{\mathcal{D}_r\}$
8:             **else if** $\text{REQVIOLATED}(\mathcal{O}_r^* \cup X_k)$ **then**
9:                 $\mathbf{D}_k^- \leftarrow \mathbf{D}_k^- \cup \{\mathcal{D}_r\}$
10:             **else**
11:                 $\mathbf{D}_k^0 \leftarrow \mathbf{D}_k^0 \cup \{\mathcal{D}_r\}$
12:         **if** $\neg\text{INCLUDESPARTITION}(\mathbf{X_D}, \langle \mathbf{D}_k^+, \mathbf{D}_k^-, \mathbf{D}_k^0 \rangle)$ **then**
13:             $\mathbf{X_D} \leftarrow \mathbf{X_D} \cup \text{MINIMIZEQUERY}(\langle X_k, \langle \mathbf{D}_k^+, \mathbf{D}_k^-, \mathbf{D}_k^0 \rangle\rangle)$
14: **return** $\mathbf{X_D}$

---

not the same partition. Finally, X_k' is added to $\mathbf{X_D}$ together with its partition $\langle \mathbf{D}_k^+, \mathbf{D}_k^-, \mathbf{D}_k^0 \rangle$. Function REQVIOLATED($arg$) returns $true$ if arg violates some requirement in R or entails some negative test case in N.

Asking the user a query X_j means asking them $(\mathcal{O}^* \models X_j?)$. Let the answering of queries by a user be modeled as function $u : \mathbf{X_D} \rightarrow \{t, f\}$. If $u_j := u(X_j) = t$, then $P \leftarrow P \cup \{X_j\}$ and $\mathbf{D} \leftarrow \mathbf{D} \setminus \mathbf{D}_j^-$. Otherwise, $N \leftarrow N \cup \{X_j\}$ and $\mathbf{D} \leftarrow \mathbf{D} \setminus \mathbf{D}_j^+$. Prospectively, according to Definition 22.2, only those diagnoses are considered in the set $\mathbf{D}$ that comply with the new DPI obtained by the addition of a test case. This allows us to formalize the problem we address in this work:

Problem Definition 22.1 (Query Selection). ***Given*** $\mathbf{D}$ *w.r.t. a DPI* $\langle \mathcal{O}, \mathcal{B}, P, N \rangle_R$, *a stopping criterion* $stop : \mathbf{D} \rightarrow \{t, f\}$ *and a user* u, ***find*** *a next query* $X_j \in \mathbf{X_D}$ *such that (1)* $(X_j, \ldots, X_q)$ *is a query sequence of minimal length and (2) there exists a* $\mathcal{D}^* \in \mathbf{D}$ *w.r.t.* $\langle \mathcal{O}, \mathcal{B}, P', N' \rangle_R$ *such that* $stop(\mathcal{D}^*) = t$, *where* $P' := P \cup \{X_i \mid X_i \in \{X_j, \ldots, X_q\}, u_i = t\}$ *and* $N' := N \cup \{X_i \mid X_i \in \{X_j, \ldots, X_q\}, u_i = f\}$.

Two strategies for selecting the "best" next query have been proposed [SFFR12]:

Split-In-Half Strategy (SPL)   selects the query X_j which minimizes the following scoring function:

$$sc_{split}(X_j) := \left| |\mathbf{D}_j^+| - |\mathbf{D}_j^-| \right| + |\mathbf{D}_j^0|$$

So, SPL prefers queries which eliminate half of the diagnoses independently of the query outcome.

Entropy-Based Strategy (ENT)   uses information about prior probabilities p_t for the user to make a mistake when using a syntactical construct of type $t \in CT(\mathcal{L})$, where $CT(\mathcal{L})$ is the set of constructors available in the used knowledge representation language $\mathcal{L}$, e.g. $\{\forall, \exists, \sqsubseteq, \neg, \sqcup, \sqcap\} \subset CT(\text{OWL DL})$. These fault probabilities p_t are assumed to be independent and used to calculate fault probabilities of axioms ax_k as

$$p(ax_k) = 1 - \prod_{t \in CT} (1 - p_t)^{n(t)}$$

where $n(t)$ is the number of occurrences of construct type t in ax_k. The probabilities of axioms can in turn be used to determine fault probabilities of diagnoses $\mathcal{D}_i \in \mathbf{D}$ as

$$p(\mathcal{D}_i) = \prod_{ax_r \in \mathcal{D}_i} p(ax_r) \prod_{ax_s \in \mathcal{O} \setminus \mathcal{D}_i} (1 - p(ax_s)). \tag{22.1}$$

ENT selects the query $X_j \in \mathbf{X_D}$ with highest expected information gain, i.e. that minimizes the following scoring function [SFFR12]:

$$sc_{ent}(X_j) = \sum_{a \in \{t,f\}} p(u_j = a) \log_2 p(u_j = a) + p(\mathbf{D}_j^0) + 1$$

where

$$p(u_j = t) = \sum_{\mathcal{D}_r \in \mathbf{D}_j^+} p(\mathcal{D}_r) + \frac{1}{2} p(\mathbf{D}_j^0)$$

and

$$p(\mathbf{D}_j^0) = \sum_{\mathcal{D}_r \in \mathbf{D}_j^0} p(\mathcal{D}_r)$$

The answer $u_j = a$ is used to update probabilities $p(\mathcal{D}_k)$ for $\mathcal{D}_k \in \mathbf{D}$ according to the Bayesian formula, yielding $p(\mathcal{D}_k | u_j = a)$.

The result of the evaluation in [SFFR12] shows that ENT reveals better performance than SPL in most of the cases. However, SPL proved to be the best strategy in situations when misleading prior information is provided, i.e. the target diagnosis $\mathcal{D}^*$ has low probability. So, one can regard ENT as a high risk strategy with high potential to perform well, depending on the priorly unknown quality of the given fault information. SPL, in contrast, can be seen as a no-risk strategy without any potential to leverage good meta information. Therefore, selection of the proper combination of prior probabilities $\{p_t \mid t \in CT(\mathcal{L})\}$ and query selection strategy is crucial for successful diagnosis discrimination and minimization of user interaction.

Example 22.2 (Example 22.1 continued) To illustrate this, let a user who wants to debug our example ontology $\mathcal{O}$ set $p(ax_i) := 0.001$ for $ax_{i(i=1,\dots,4)}$ and $p(ax_5) := 0.1, p(ax_6) := 0.15$, e.g. because the user doubts the correctness of ax_5, ax_6 while being quite sure that $ax_{i(i=1,\dots,4)}$ are correct. Assume that $\mathcal{D}_2$ corresponds to the target diagnosis $\mathcal{D}^*$, i.e. the settings provided by the user are inept. Application of ENT starts with computation of prior fault probabilities of diagnoses $p(\mathcal{D}_1) = p(\mathcal{D}_2) = p(\mathcal{D}_3) = p(\mathcal{D}_4) = 0.003, p(\mathcal{D}_5) = 0.393, p(\mathcal{D}_6) = 0.591$ (Formula 22.1). Then $(\mathcal{O}^* \models X_1?)$ with $X_1 := \{DeptEmployee(s), Student(s)\}$, will be identified as the optimal query since it has the minimal score $sc_{ent}(X_1) = 0.02$ (see Table 22.1 for queries and partitions w.r.t. the example ontology). However, since the unfavorable answer $u_1 = f$ is given, this query eliminates only two of six diagnoses $\mathcal{D}_4$ and $\mathcal{D}_6$. The Bayesian probability update then yields $p(\mathcal{D}_2) = p(\mathcal{D}_3) = p(\mathcal{D}_4) = 0.01$ and $p(\mathcal{D}_5) = 0.97$. As next query X_2 with $sc_{ent}(X_2) = 0.811$ is selected and answered unfavorably ($u_2 = t$) as well which results in the elimination of only one of four diagnoses $\mathcal{D}_5$. By querying X_3 ($sc_{ent}(X_3) = 0.082, u_3 = t$) and X_4 ($sc(X_4) = 0, u_4 = t$), the further execution of this procedure finally leads to the target diagnosis $\mathcal{D}_2$. So, application of ENT requires four queries to find $\mathcal{D}^*$. If SPL is used instead, only three queries are required. The algorithm can select one of the two queries X_5 or X_9 because each eliminates half of all diagnoses in any case. Let the strategy select X_5 which is answered positively ($u_5 = t$). As successive queries, X_6 ($u_6 = f$) and X_1 ($u_1 = f$) are selected, which leads to the revelation of $\mathcal{D}^* = \mathcal{D}_2$. $\qquad\square$

Query	$\mathbf{D}_i^+$	$\mathbf{D}_i^-$	$\mathbf{D}_i^0$
$X_1 : \{DeptEmployee(s), Student(s)\}$	$\mathcal{D}_4, \mathcal{D}_6$	$\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_5$	$\emptyset$
$X_2 : \{PhD(s)\}$	$\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_6$	$\mathcal{D}_5$	$\emptyset$
$X_3 : \{Researcher(s)\}$	$\mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_6$	$\mathcal{D}_1, \mathcal{D}_5$	$\emptyset$
$X_4 : \{Student(s)\}$	$\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_4, \mathcal{D}_5, \mathcal{D}_6$	$\mathcal{D}_3$	$\emptyset$
$X_5 : \{Researcher(s), Student(s)\}$	$\mathcal{D}_2, \mathcal{D}_4, \mathcal{D}_6$	$\mathcal{D}_1, \mathcal{D}_3, \mathcal{D}_5$	$\emptyset$
$X_6 : \{DeptMember(s)\}$	$\mathcal{D}_3, \mathcal{D}_4$	$\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_5, \mathcal{D}_6$	
$X_7 : \{PhD(s), Student(s)\}$	$\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_4, \mathcal{D}_6$	$\mathcal{D}_3, \mathcal{D}_5$	$\emptyset$
$X_8 : \{DeptMember(s), Student(s)\}$	$\mathcal{D}_2$	$\mathcal{D}_1, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5, \mathcal{D}_6$	$\emptyset$
$X_9 : \{DeptEmployee(s)\}$	$\mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_6$	$\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_5$	$\emptyset$

Table 22.1: A set $\mathbf{X_D}$ of queries and associated partitions w.r.t. the initial DPI $\langle \mathcal{T}, \mathcal{A}, \emptyset, \emptyset \rangle_{\{\text{coherence}\}}$ of the example ontology $\mathcal{O}$.

This scenario demonstrates that the no-risk strategy SPL (*three queries*) is more suitable than ENT (*four queries*) for fault probabilities which disfavor the target diagnosis. Let us suppose, on the other hand, that probabilities are assigned more reasonably in our example, e.g. $\mathcal{D}^* = \mathcal{D}_6$. Then it will take ENT only *two queries* (X_1, X_6) to find $\mathcal{D}^*$ while SPL will still require *three queries*, e.g. (X_5, X_1, X_6).

This example indicates that, unless the target diagnosis is known in advance, one can never be sure to select the best strategy from SPL and ENT. In Chapte 23 we present a learning query selection algorithm that combines the benefits of both SPL and ENT. It adapts the way of selecting the next query depending on the elimination rate (like SPL) and on information gain (like ENT). Thereby its performance approaches the performance of the better of both SPL and ENT.

Chapter 23

RIO: Risk Optimization for Query Selection

The proposed Risk Optimization Algorithm (RIO) extends ENT strategy with a dynamic learning procedure that learns by reinforcement how to select the next query. Its behavior is determined by the achieved performance in terms of diagnosis elimination rate w.r.t. the set of leading diagnoses $\mathbf{D}$. Good performance causes similar behavior to ENT, whereas aggravation of performance leads to a gradual neglect of the given meta information, and thus to a behavior akin to SPL. Like ENT, RIO continually improves the prior fault probabilities based on new knowledge obtained through queries to a user.

RIO learns a "cautiousness" parameter c whose admissible values are captured by the user-defined interval $[\underline{c}, \overline{c}]$. The relationship between c and queries is as follows:

Definition 23.1 (Cautiousness of a Query). *We define the* cautiousness $c_q(X_i)$ *of a query* $X_i \in \mathbf{X_D}$ *as follows:*

$$c_q(X_i) := \frac{\min\left\{|\mathbf{D}_i^+|, |\mathbf{D}_i^-|\right\}}{|\mathbf{D}|} \in \left[0, \frac{\left\lfloor \frac{|\mathbf{D}|}{2} \right\rfloor}{|\mathbf{D}|}\right] =: [\underline{c_q}, \overline{c_q}]$$

A query X_i *is called* braver *than query* X_j *iff* $c_q(X_i) < c_q(X_j)$. *Otherwise* X_i *is called* more cautious *than* X_j. *A query with maximum cautiousness* $\overline{c_q}$ *is called* no-risk query.

Definition 23.2 (Elimination Rate). *Given a query* X_i *and the corresponding answer* $u_i \in \{t, f\}$, *the elimination rate*

$$e(X_i, u_i) = \frac{|\mathbf{D}_i^-|}{|\mathbf{D}|} \qquad \text{if } u_i = t$$

and

$$e(X_i, u_i) = \frac{|\mathbf{D}_i^+|}{|\mathbf{D}|} \qquad \text{if } u_i = f$$

The answer u_i *to a query* X_i *is called* favorable *iff it maximizes the elimination rate* $e(X_i, u_i)$. *Otherwise* u_i *is called* unfavorable. *The minimal or* worst case *elimination rate* $\min_{u_i \in \{t,f\}}(e(X_i, u_i))$ *of* X_i *is denoted by* $e_{wc}(X_i)$.

So, the cautiousness $c_q(X_i)$ of a query X_i is exactly the worst case elimination rate, i.e. $c_q(X_i) = e_{wc}(X_i) = e(X_i, u_i)$ given that u_i is the unfavorable query result. Intuitively, parameter c characterizes the minimum proportion of diagnoses in $\mathbf{D}$ which should be eliminated by the successive query.

Definition 23.3 (High-Risk Query). *Given a query X_i and cautiousness c, X_i is called a* high-risk query *iff $c_q(X_i) < c$, i.e. the cautiousness of the query is lower than the algorithm's current cautiousness value c. Otherwise, X_i is called* non-high-risk query. *By $NHR_c(\mathbf{X_D}) \subseteq \mathbf{X_D}$ we denote the set of non-high-risk queries w.r.t. c. For given cautiousness c, the set of queries $\mathbf{X_D}$ can be partitioned in high-risk queries and non-high-risk queries.*

Example 23.1 (Example 22.2 continued) Let the user specify $c := 0.3$ for the set $\mathbf{D}$ with $|\mathbf{D}| = 6$. Given these settings, $X_1 := \{DeptEmployee(s), Student(s)\}$ is a non-high-risk query since its partition $\langle \mathbf{D}_1^+, \mathbf{D}_1^-, \mathbf{D}_1^0 \rangle = \langle \{\mathcal{D}_4, \mathcal{D}_6\}, \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_5\}, \emptyset \rangle$ and thus its cautiousness $c_q(X_1) = 2/6 \geq 0.3 = c$. The query $X_2 := \{PhD(s)\}$ with the partition $\langle \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_6\}, \{\mathcal{D}_5\}, \emptyset \rangle$ is a high-risk query because $c_q(X_2) = 1/6 < 0.3 = c$ and $X_3 := \{Researcher(s), Student(s)\}$ with $\langle \{\mathcal{D}_2, \mathcal{D}_4, \mathcal{D}_6\}, \{\mathcal{D}_1, \mathcal{D}_3, \mathcal{D}_5\}, \emptyset \rangle$ is a no-risk query due to $c_q(X_3) = 3/6 = \overline{c_q}$. □

Given a user's answer u_s to a query X_s, the cautiousness c is updated depending on the elimination rate $e(X_s, u_s)$ by $c \leftarrow c + c_{adj}$ where the cautiousness adjustment factor $c_{adj} := 2(\overline{c} - \underline{c})adj$. The scaling factor $2(\overline{c} - \underline{c})$ regulates the extent of the cautiousness adjustment depending on the interval length $\overline{c} - \underline{c}$. More crucial is the factor adj that indicates the sign and magnitude of the cautiousness adjustment:

$$adj := \frac{\left\lfloor \frac{|\mathbf{D}|}{2} - \varepsilon \right\rfloor}{|\mathbf{D}|} - e(X_s, u_s)$$

where $\varepsilon \in (0, \frac{1}{2})$ is a constant which prevents the algorithm from getting stuck in a no-risk strategy for even $|\mathbf{D}|$. E.g., given $c = 0.5$ and $\varepsilon = 0$, the elimination rate of a no-risk query $e(X_s, u_s) = \frac{1}{2}$ resulting always in $adj = 0$. The value of ε can be set to an arbitrary real number, e.g. $\varepsilon := \frac{1}{4}$. If $c + c_{adj}$ is outside the user-defined cautiousness interval $[\underline{c}, \overline{c}]$, it is set to $\underline{c}$ if $c < \underline{c}$ and to $\overline{c}$ if $c > \overline{c}$. Positive c_{adj} is a penalty telling the algorithm to get more cautious, whereas negative c_{adj} is a bonus resulting in a braver behavior of the algorithm. Note, for the user-defined interval $[\underline{c}, \overline{c}] \subseteq [c_q, \overline{c_q}]$ must hold. $\underline{c} - c_q$ and $\overline{c_q} - \overline{c}$ represent the minimal desired difference in performance to a high-risk (ENT) and no-risk (SPL) query selection, respectively. By expressing trust (disbelief) in the prior fault probabilities through specification of lower (higher) values for $\underline{c}$ and/or $\overline{c}$, the user can take influence on the behavior of RIO.

Example 23.2 (Example 23.1 continued) Assume $p(ax_i) := 0.001$ for $ax_{i(i=1,...,4)}$ and $p(ax_5) := 0.1$, $p(ax_6) := 0.15$ and the user rather disbelieves these fault probabilities and thus sets $c = 0.4$, $\underline{c} = 0$ and $\overline{c} = 0.5$. In this case RIO selects a no-risk query X_3 just as SPL would do. Given $u_3 = t$ and $|\mathbf{D}| = 6$, the algorithm computes the elimination rate $e(X_3, t) = 0.5$ and adjusts the cautiousness by $c_{adj} = -0.17$ which yields $c = 0.23$. This allows RIO to select a higher-risk query in the next iteration, whereupon the target diagnosis $\mathcal{D}^* = \mathcal{D}_2$ is found after asking three queries. In the same situation, ENT (starting with high-risk query X_1) would require four queries. □

RIO, described in Algorithm 14, starts with the computation of minimal diagnoses. GETDIAGNOSES function implements a combination of HS-Tree and QuickXPlain algorithms [SFFR12]. Using uniform-cost search, the algorithm extends the set of leading diagnoses $\mathbf{D}$ with a maximum number of most probable minimal diagnoses such that $|\mathbf{D}| \leq n$.

Then the GETPROBABILITIES function calculates the fault probabilities $p(\mathcal{D}_i)$ for each diagnosis $\mathcal{D}_i$ of the set of leading diagnoses $\mathbf{D}$ using Formula (22.1). Next it adjusts the probabilities as per the Bayesian theorem taking into account all previous query answers which are stored in P and N. Finally, the resulting probabilities $p_{adj}(\mathcal{D}_i)$ are normalized. Based on the set of leading diagnoses $\mathbf{D}$,

Algorithm 14 Risk Optimization Algorithm (RIO)

Input: DPI $\langle \mathcal{O}, \mathcal{B}, P, N \rangle_R$, fault probabilities of diagnoses DP, cautiousness $C = (c, \underline{c}, \overline{c})$, number of leading diagnoses n to be considered, acceptance threshold σ
Output: a minimal diagnosis $\mathcal{D}$ w.r.t. $\langle \mathcal{O}, \mathcal{B}, P, N \rangle_R$

1: $\mathbf{D} \leftarrow \emptyset$
2: **repeat**
3:     $\mathbf{D} \leftarrow \text{GETDIAGNOSES}(\mathbf{D}, n, \mathcal{O}, \mathcal{B}, P, N)$
4:     $DP \leftarrow \text{GETPROBABILITIES}(DP, \mathbf{D}, P, N)$
5:     $\mathbf{X_D} \leftarrow \text{GENERATEQUERIES}(\mathcal{O}, \mathcal{B}, P, \mathbf{D})$
6:     $X_s \leftarrow \text{GETMINSCOREQUERY}(DP, \mathbf{X_D})$
7:     **if** $\text{GETQUERYCAUTIOUSNESS}(X_s, \mathbf{D}) < c$ **then**
8:         $X_s \leftarrow \text{GETALTERNATIVEQUERY}(c, \mathbf{X_D}, DP, \mathbf{D})$
9:     **if** $\text{GETANSWER}(X_s) = \textit{yes}$ **then**
10:         $P \leftarrow P \cup \{X_s\}$
11:     **else**
12:         $N \leftarrow N \cup \{X_s\}$
13:     $c \leftarrow \text{UPDATECAUTIOUSNESS}(\mathbf{D}, P, N, X_s, c, \underline{c}, \overline{c})$
14: **until** $\text{ABOVETHRESHOLD}(DP, \sigma) \vee \text{ELIMINATIONRATE}(X_s) = 0$
15: **return** $\text{MOSTPROBABLEDIAG}(\mathbf{D}, DP)$

GENERATEQUERIES generates queries according to Algorithm 13. GETMINSCOREQUERY determines the best query $X_{sc} \in \mathbf{X_D}$ according to sc_{ent}:

$$X_{sc} = \underset{X_k \in \mathbf{X_D}}{\arg \min}(sc_{ent}(X_k))$$

If X_{sc} is a non-high-risk query, i.e. $c \leq c_q(X_{sc})$ (determined by GETQUERYCAUTIOUSNESS), X_{sc} is selected. In this case, X_{sc} is the query with best information gain in $\mathbf{X_D}$ and moreover guarantees the required elimination rate specified by c.

Otherwise, GETALTERNATIVEQUERY selects the query $X_{alt} \in \mathbf{X_D}$ $(X_{alt} \neq X_{sc})$ which has minimal score sc_{ent} among all least cautious non-high-risk queries L_c. That is,

$$X_{alt} = \underset{X_k \in L_c}{\arg \min}(sc_{ent}(X_k))$$

where

$$L_c := \{X_r \in NHR_c(\mathbf{X_D}) \mid \forall X_t \in NHR_c(\mathbf{X_D}) : c_q(X_r) \leq c_q(X_t)\}$$

If there is no such query $X_{alt} \in \mathbf{X_D}$, then X_{sc} is selected.

Given the user's answer u_s, the selected query $X_s \in \{X_{sc}, X_{alt}\}$ is added to P or N accordingly (see Chapter 22). In the last step of the main loop the algorithm updates the cautiousness value c (function UPDATECAUTIOUSNESS) as described above.

Before the next query selection iteration starts, a stop condition test is performed. The algorithm evaluates whether the most probable diagnosis is at least $\sigma\%$ more likely than the second most probable diagnosis (ABOVETHRESHOLD) or none of the leading diagnoses has been eliminated by the previous query, i.e. GETELIMINATIONRATE returns zero for X_s. If a stop condition is met, the presently most likely diagnosis is returned (MOSTPROBABLEDIAG).

Chapter 24

Evaluation

Goals. This evaluation should demonstrate that (1) there is a significant discrepancy between existing strategies SPL and ENT concerning user effort where the winner depends on the quality of meta information, (2) RIO exhibits superior average behavior compared to ENT and SPL w.r.t. the amount of user interaction required, irrespective of the quality of specified fault information, (3) RIO scales well and (4) its reaction time is well suited for an interactive debugging approach.

Provenance of Test Data. As data source for the evaluation we used faulty real-world ontologies produced by automatic ontology matching systems (cf. Example 22.1). Matching of two ontologies $\mathcal{O}_i$ and $\mathcal{O}_j$ is understood as detection of correspondences between elements of these ontologies [SE13]:

Definition 24.1 (Ontology matching). *Let $Q(\mathcal{O}) \subseteq \mathbf{S}(\mathcal{O})$ denote the set of matchable elements in an ontology $\mathcal{O}$, where $\mathbf{S}(\mathcal{O})$ denotes the signature of $\mathcal{O}$. An ontology matching operation determines an alignment $\mathcal{M}_{ij}$, which is a set of correspondences between matched ontologies $\mathcal{O}_i$ and $\mathcal{O}_j$. Each* correspondence *is a 4-tuple $\langle x_i, x_j, r, v \rangle$, such that $x_i \in Q(\mathcal{O}_i)$, $x_j \in Q(\mathcal{O}_j)$, r is a semantic relation and $v \in [0,1]$ is a confidence value. We call $\mathcal{O}_{i\mathcal{M}j} := \mathcal{O}_i \cup \phi(\mathcal{M}_{ij}) \cup \mathcal{O}_j$ the aligned ontology for $\mathcal{O}_i$ and $\mathcal{O}_j$ where ϕ maps each correspondence to an axiom.*

Let in the following $Q(\mathcal{O})$ be the restriction to atomic concepts and roles in $\mathbf{S}(\mathcal{O})$, $r \in \{\sqsubseteq, \sqsupseteq, \equiv\}$ and ϕ the natural alignment semantics [MS09] that maps correspondences one-to-one to axioms of the form $x_i \, r \, x_j$. We evaluate RIO using aligned ontologies by the following reasons: (1) Matching results often cause inconsistency/incoherence of ontologies. (2) The (fault) structure of different ontologies obtained through matching generally varies due to different authors and matching systems involved in the genesis of these ontologies. (3) For the same reasons, it is hard to estimate the quality of fault probabilities, i.e. it is unclear which of the existing query selection strategies to chose for best performance. (4) Available reference mappings can be used as correct solutions of the debugging procedure.

Test Datasets. We used two datasets D1 and D2: Each faulty aligned ontology $\mathcal{O}_{i\mathcal{M}j}$ in D1 is the result of applying one of four ontology matching systems to a set of six independently created ontologies in the domain of conference organization. For a given pair of ontologies $\mathcal{O}_i \neq \mathcal{O}_j$, each system produced an alignment $\mathcal{M}_{ij}$. The average size of $\mathcal{O}_{i\mathcal{M}j}$ per matching system was between 312 and 377 axioms. D1 is a superset of the dataset used in [Stu08] for which all debugging systems under evaluation manifested correctness or scalability problems. D2, used to assess the scalability of RIO, is the set of ontologies from the ANATOMY track in the Ontology Alignment Evaluation Initiative[38] (OAEI) 2011.5 [SE13], which

---

[38]http://oaei.ontologymatching.org

comprises two input ontologies $\mathcal{O}_1$ (11545 axioms) and $\mathcal{O}_2$ (4838 axioms). The size of the aligned ontologies generated by results of seven different matching systems was between 17530 and 17844 axioms.[39]

Reference Solutions.   For the dataset D1, based on a manually produced reference alignment $\mathcal{R}_{ij} \subseteq \mathcal{M}_{ij}$ for ontologies $\mathcal{O}_i, \mathcal{O}_j$ (cf. [MST08]), we were able to fix a target diagnosis $\mathcal{D}^* := \phi(\mathcal{M}_{ij} \setminus \mathcal{R}_{ij})$ for each incoherent $\mathcal{O}_{i\mathcal{M}j}$. In cases where $\mathcal{D}^*$ represented a non-minimal diagnosis, it was randomly redefined as a minimal diagnosis $\mathcal{D}^* \subset \phi(\mathcal{M}_{ij} \setminus \mathcal{R}_{ij})$. In case of D2, given the ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$, the output $\mathcal{M}_{12}$ of a matching system, and the correct reference alignment $\mathcal{R}_{12}$, we fixed $\mathcal{D}^*$ as follows: We carried out (prior to the actual experiment) a debugging session with DPI $\langle \phi(\mathcal{M}_{12} \setminus \mathcal{R}_{12}),$ $\mathcal{O}_1 \cup \mathcal{O}_2 \cup \phi(\mathcal{M}_{12} \cap \mathcal{R}_{12}), \emptyset, \emptyset \rangle_{\{\text{coherence}\}}$ and randomly chose one of the identified diagnoses as $\mathcal{D}^*$.

Test Settings.   We conducted 4 experiments EXP-i ($i = 1, \ldots, 4$), the first two with dataset D1 and the other two with D2. In experiments 1 and 3 we simulated good fault probabilities by setting $p(ax_k) :=$ 0.001 for $ax_k \in \mathcal{O}_i \cup \mathcal{O}_j$ and $p(ax_m) := 1 - v_m$ for $ax_m \in \mathcal{M}_{ij}$, where v_m is the confidence of the correspondence underlying ax_m. Unreasonable fault information was used in experiments 2 and 4. In EXP-4 the following probabilities were defined: $p(ax_k) := 0.01$ for $ax_k \in \mathcal{O}_i \cup \mathcal{O}_j$ and $p(ax_m) :=$ 0.001 for $ax_m \in \mathcal{M}_{ij}$. In EXP-2, in contrast, we used probability settings of EXP-1, but altered the target diagnosis $\mathcal{D}^*$ in that we precomputed (before the actual experiment started) the 30 most probable minimal diagnoses, and from these we selected the diagnosis with the highest number of axioms $ax_k \in \mathcal{O}_{i\mathcal{M}j} \setminus \phi(\mathcal{M}_{ij})$ as $\mathcal{D}^*$.

Throughout all four experiments, we set $|\mathbf{D}| := 9$ (which proved to be a good trade-off between computation effort and representativeness of the leading diagnoses), $\sigma := 85\%$ and as input parameters for RIO we set $c := 0.25$ and $[\underline{c}, \overline{c}] := [c_{\min}, c_{\max}] = [0, \frac{4}{9}]$. To let tests constitute the highest challenge for the evaluated methods, the initial DPI was specified as $\langle \mathcal{O}_{i\mathcal{M}j}, \emptyset, \emptyset, \emptyset \rangle_{\{\text{coherence}\}}$, i.e. the entire search space was explored without adding parts of $\mathcal{O}_{i\mathcal{M}j}$ to $\mathcal{B}$, although $\mathcal{D}^*$ was always a subset of the alignment $\mathcal{M}_{ij}$ only. In practice, given such prior knowledge, the search space could be severely restricted and debugging greatly accelerated. All tests were executed on a Core-i7 (3930K) 3.2Ghz, 32GB RAM with Ubuntu Server 11.04 and Java 6 installed.[40]

Metrics.   Each experiment involved a debugging session of ENT, SPL as well as RIO for each ontology in the respective dataset. In each debugging run we measured the number of required queries (q) until $\mathcal{D}^*$ was identified, the overall debugging time (*debug*) assuming that queries are answered instantaneously and the reaction time (*react*), i.e. the average time between two successive queries. The queries generated in the tests were answered by an automatic oracle by means of the target ontology $\mathcal{O}_{i\mathcal{M}j} \setminus \mathcal{D}^*$.

Observations.   The difference w.r.t. the number of queries per test run between the better and the worse strategy in {SPL,ENT} was absolutely significant, with a maximum of 2300% in EXP-4 and averages of 190% to 1145% throughout all four experiments (Figure 24.2). Moreover, results show that varying quality of fault probabilities in {EXP-1,EXP-3} compared to {EXP-2,EXP-4} clearly affected the performance of ENT and SPL (see first two rows in Figure 24.2). This perfectly motivates the application of RIO.

Results of both experimental sessions, $\langle$EXP-1,EXP-2$\rangle$ and $\langle$EXP-3,EXP-4$\rangle$, are summarized in Figures 24.1(a) and 24.1(b), respectively. The figures show the (average) number of queries asked by RIO

---

[39]Source ontologies, produced alignments by each matcher, and reference alignments were downloaded from http://bit.ly/Zffkow (D1) and http://bit.ly/Koh1NB as well as http://bit.ly/MU5Ca9 (D2).

[40]See http://code.google.com/p/rmbd/wiki for code and details.

	EXP-1			EXP-2			EXP-3			EXP-4		
	debug	*react*	*q*	*debug*	*react*	*q*	*debug*	*react*	*q*	*debug*	*react*	*q*
ENT	1860	262	3.67	1423	204	5.26	**60928**	12367	5.86	74463	5629	11.86
SPL	**1427**	**159**	5.70	**1237**	**148**	5.44	104910	**4786**	19.43	98647	**4781**	18.29
RIO	1592	286	**3.00**	1749	245	**4.37**	62289	12825	**5.43**	**66895**	8327	**8.14**

Table 24.1: Average time (ms) for the entire debugging session (*debug*), average time (ms) between two successive queries (*react*), and average number of queries (*q*) required by each strategy.
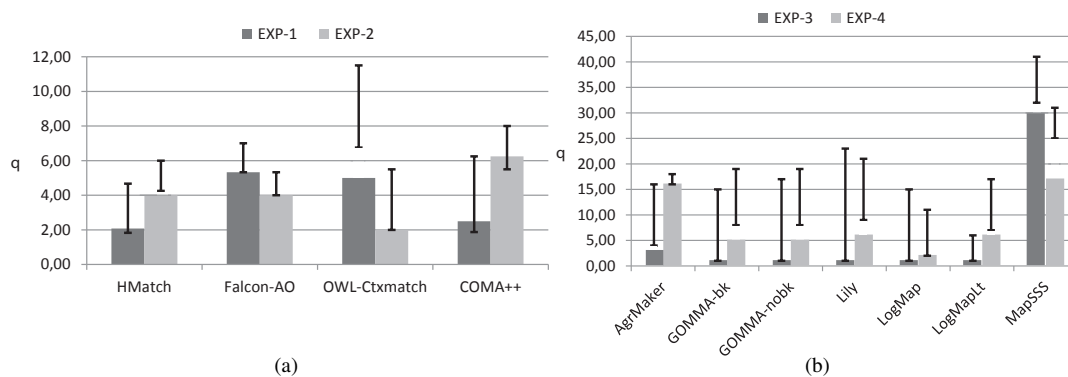

(a)   (b)

Figure 24.1: The bars show the avg. number of queries (q) needed by RIO, grouped by matching tools. The distance from the bar to the lower (upper) end of the whisker indicates the avg. difference of RIO to the queries needed by the per-session better (worse) strategy of SPL and ENT, respectively.

and the (average) differences to the number of queries needed by the per-session better and worse strategy in {SPL,ENT}, respectively. The results illustrate clearly that the average performance achieved by RIO was always substantially closer to the better than to the worse strategy. In both EXP-1 and EXP-2, throughout 74% of 27 debugging sessions, RIO worked as efficiently as the best strategy (Figure 24.2). In 26% of the cases in EXP-2, RIO even outperformed both other strategies; in these cases, RIO could save more than 20% of user interaction on average compared to the best other strategy. In one scenario in EXP-1, it took ENT 31 and SPL 13 queries to finish, whereas RIO required only 6 queries, which amounts to an improvement of more than 80% and 53%, respectively. In ⟨EXP-3,EXP-4⟩, the savings achieved by RIO were even more substantial. RIO manifested superior behavior to both other strategies in 29% and 71% of cases, respectively. Not less remarkable, in 100% of the tests in EXP-3 and EXP-4, RIO was at least as efficient as the best other strategy. Recalling Figure 24.2, this means that RIO can

	EXP-1	EXP-2	EXP-3	EXP-4
$q_{SPL} < q_{ENT}$	11%	37%	0%	29%
$q_{ENT} < q_{SPL}$	81%	56%	100%	71%
$q_{SPL} = q_{ENT}$	7%	7%	0%	0%
$q_{RIO} < \min$	4%	26%	29%	71%
$q_{RIO} \leq \min$	74%	74%	100%	100%

Table 24.2: Percentage rates indicating which strategy performed best/better w.r.t. the required user interaction, i.e. number of queries. EXP-1 and EXP-2 involved 27, EXP-3 and EXP-4 seven debugging sessions each. q_{str} denotes the number of queries needed by strategy str and $\min$ is an abbreviation for $\min(q_{SPL}, q_{ENT})$.
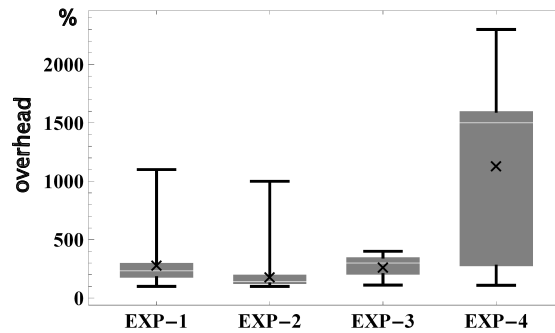
Figure 24.2: Box-Whisker Plots presenting the distribution of overhead $(q_w - q_b)/q_b * 100$ (in %) per debugging session of the worse strategy $q_w := \max(q_{\text{SPL}}, q_{\text{ENT}})$ compared to the better strategy $q_b := \min(q_{\text{SPL}}, q_{\text{ENT}})$. Mean values are depicted by a cross.

avoid query overheads of over 2000%. Table 24.1, which provides average values for q, *react* and *debug* per strategy, demonstrates that RIO is the best choice in all experiments w.r.t. q. Consequently, RIO is suitable for both good and poor meta information.

As to time aspects, RIO manifested good performance, too. Since times consumed in ⟨EXP-1,EXP-2⟩ are almost negligible, consider the more meaningful results obtained in ⟨EXP-3,EXP-4⟩. While the best reaction time in both experiments was achieved by SPL, we can clearly see that SPL was significantly inferior to both ENT and RIO concerning q and *debug*. RIO revealed the best debugging time in EXP-4, and needed only 2.2% more time than the best strategy (ENT) in EXP-3. However, if we assume the user being capable of reading and answering a query in, e.g., 30 sec on average, which is already quite fast, then the overall time savings of RIO compared to ENT in EXP-3 would already account for 5%. Doing the same thought experiment for EXP-4, RIO would save 25% (w.r.t. ENT) and 50% (w.r.t. SPL) of debugging time on average. All in all, the measured times confirm that RIO is well suited for *interactive* debugging.

Chapter 25

Related Work

A similar interactive technique was presented in [NRG12], where a user is successively asked single ontology axioms in order to obtain a partition of a given ontology into a set of desired and a set of undesired consequences. However, given an inconsistent/incoherent ontology, this technique starts from an empty set of desired consequences aiming at adding to this set only axioms which preserve coherence, whereas our approach starts from the complete ontology aiming at finding a minimal set of axioms responsible for the violation of pre-specified requirements.

An approach for alignment debugging was proposed in [Mei11]. This work describes approximate algorithms for computing a "local optimal diagnosis" and complete methods to discover a "global optimal diagnosis". Optimality in this context refers to the maximum sum of confidences in the resulting coherent alignment. In contrast to our framework, diagnoses are determined automatically without support for user interaction. Instead, techniques for manual revision of the alignment as a procedure *independent* from debugging are demonstrated.

Chapter 26

Summary and Conclusions

We have shown problems of state-of-the-art interactive ontology debugging strategies w.r.t. the usage of unreliable meta information. To tackle this issue, we proposed a learning strategy which combines the benefits of existing approaches, i.e. high potential and low risk. Depending on the performance of the diagnosis discrimination actions, the trust in the a-priori information is adapted. Tested under various conditions, our algorithm revealed good scalability and reaction time as well as superior average performance to two common approaches in the field in all tested cases w.r.t. required user interaction. Highest achieved savings amounted to more than 80% and user interaction overheads resulting from the wrong choice of strategy of up to 2300% could be saved. In the hardest test cases, the new strategy was not only on average, but in 100% of the test cases at least as good as the best other strategy.

Part VI

A Direct Approach to Sequential Diagnosis of High Cardinality Faults in Knowledge Bases

In this part we cover the topic of efficiently dealing with KB debugging problems involving high cardinality faults. This part relies on material [SFRF14c, SFRF14a, SFRF14b] published in the *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, in *DX 2014 - 25th International Workshop on Principles of Diagnosis* and in the *Proceedings of the Third International Workshop on Debugging Ontologies and Ontology Mappings (WoDOOM14)*, respectively.[41]

---

[41]We are glad to report that the publication [SFRF14a] was awarded the Best Paper Award at the DX Workshop that took place in Graz, Austria in September 2014 (see http://dx-2014.ist.tugraz.at).

Chapter 27

Introduction to the Problem

Model-based diagnosis (MBD) [Rei87] is a general method which can be used to find errors in hardware, software, knowledge-bases (KBs), orchestrated web-services, configurations, etc. In particular, ontology (KB) debugging tools [KPHS07, FS05, HPS08] can localize a (potential) fault by finding sets of axioms $\mathcal{D} \subseteq \mathcal{K}$ called *diagnoses* for the KB $\mathcal{K}$. Diagnoses are generated using minimal conflict sets, i.e. irreducible sets of axioms $CS \subseteq \mathcal{K}$ that violate some requirements, by using a consistency checker (black-box approach). At least all axioms of a minimal diagnosis must be modified or deleted in order to formulate a fault-free knowledge-base $\mathcal{K}^*$. A knowledge-base $\mathcal{K}$ is faulty if some requirements, such as consistency of $\mathcal{K}$, presence or absence of specific entailments, are violated.

Sequential MBD methods [dKW87] applied to KB debugging acquire additional information in order to discriminate between diagnoses [SFFR12]. Generated queries are answered by some oracle providing additional observations about the entailments of a valid KB. As various applications show, the standard methods work very satisfactorily for cases where the number of faults (minimal conflict sets) is low (single digit number), consistency checking is fast (single digit number of seconds), and sufficient possibilities for observations are available.

However, there are situations when KBs comprise a large number of faults. For example, in ontology matching scenarios two KBs with several thousands of axioms are merged into a single one. High quality matchers (e.g. [JRG11]) require the diagnosis of such substantially extended KBs, but could not apply standard diagnosis methods because of the large number of minimal diagnoses and their high cardinality. E.g. there are cases when the minimum cardinality of diagnoses is greater than 20.

In order to deal with hard diagnosis instances, we propose to relax the requirement for sequential diagnosis to compute a set of *preferred* minimal diagnoses, such as a set of most probable diagnoses. Instead, we compute just *some* set of minimal diagnoses which can be used for query generation. This allows to use direct computation of diagnoses [SU06] *without* computing conflict sets. The direct approach was applied for non-interactive diagnosis of ontologies [DQPS11, BKP12] and constraints [FSZ11]. A recent approach [SKFP12] does not generate the standard HS-TREE, but still depends on the minimization of conflict sets, i.e. $|\mathcal{D}|$ minimized conflicts have to be discovered. Consequently, if $|\mathcal{D}| \gg m$, substantially more consistency checks are required, where $|\mathcal{D}|$ is the cardinality of the minimal diagnosis and m is the number of minimal diagnoses required for query generation.

Since we are replacing the set of most probable diagnoses by just a set of minimal diagnoses, some important practical questions have to be addressed. (1) Is a substantial number of additional queries needed, (2) is this approach able to locate the faults, and (3) how efficient is this approach?

In order to answer these questions we have exploited the most difficult diagnosis problems of the ontology alignment competition [EFvH$^+$11]. Our evaluation shows that sequential diagnosis by direct diagnosis generation needs approximately the same number of queries (± 1) in order to identify the faults.

This evaluation was carried out for cases where the standard sequential diagnosis method was applicable. Furthermore, the evaluation shows that our proposed method is able to locate faults in all cases correctly, particularly in those cases where debugging sessions by means of the standard method are not successful (due to overwhelming time or space consumption). Moreover, for the hardest cases (i.e., more than 4 minutes overall debugging time), the additional computation costs introduced by the direct method apart from the costs needed for theorem proving are less than 50%, i.e. reasoning costs amount to more than two thirds of overall computation time.

The rest of Part VI is organized as follows: Chapter 28 gives a brief introduction to the main notions of sequential KB diagnosis. The details of the suggested algorithms are presented in Chapter 29. In Chapter 30 we provide evaluation results whereupon Chapter 31 gives a conclusion.

Chapter 28

Basic Concepts

In the following we present (1) the fundamental concepts regarding the diagnosis of KBs and (2) the interactive localization of axioms which must be changed.

Diagnosis of KBs. Given a knowledge-base $\mathcal{K}$ which is a set of logical sentences (axioms), the user can specify particular requirements during the knowledge-engineering process. The most basic requirement is satisfiability, i.e. a logical model exists. A further frequently employed requirement is coherence. Coherence requires that there exists a model s.t. the interpretation of every unary predicate is non-empty. In other words, if we add $\exists Y\, a(Y)$ to $\mathcal{K}$ for every unary predicate a, then the resulting KB must be satisfiable. In addition, as it is common practice in software engineering, the knowledge-engineer (user for short) may specify test cases. Test cases are axioms which must (not) be entailed by a valid KB.

Definition 28.1. *Given a set of axioms P (called positive test cases) and a set of axioms N (called negative test cases), a knowledge-base $\mathcal{K}^*$ is* valid *iff it fulfills the following requirements:*

1. *$\mathcal{K}^*$ is satisfiable (and coherent if required)*

2. *$\mathcal{K}^* \models p \quad \forall p \in P$*

3. *$\mathcal{K}^* \not\models n \quad \forall n \in N$*

Let us assume that there is a non-valid KB $\mathcal{K}$, then a set of axioms $\mathcal{D} \subseteq \mathcal{K}$ must be removed and possibly some axioms EX must be added by the user s.t. an updated $\mathcal{K}^*$ becomes valid, i.e. $\mathcal{K}^* := (\mathcal{K} \setminus \mathcal{D}) \cup EX$. The goal of diagnosis is to provide information to the users which are the sets of axioms $\mathcal{D}$ (which is called a diagnosis) that must be changed. In order to prevent unnecessary changes, $\mathcal{D}$ is often required to be subset-minimal, i.e. the set should be as small as possible. Furthermore, we allow the user to define a set of axioms $\mathcal{B}$ (called the background theory) which must not be changed (i.e. the correct axioms). More formally:

Definition 28.2. *Given a diagnosis problem instance (DPI) specified by $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ where*

- *$\mathcal{K}$ is a knowledge-base,*

- *$\mathcal{B}$ a background theory,*

- *P a set of axioms which must be implied by a valid knowledge-base $\mathcal{K}^*$ and*

- *N a set of axioms, each of which must* not *be implied by $\mathcal{K}^*$*

$\mathcal{D} \subseteq \mathcal{K}$ *is a* diagnosis *w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ *iff* $\mathcal{K} \setminus \mathcal{D}$ *can be extended by a set of logical sentences* EX *such that:*

1. $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX$ *is consistent*

2. $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \models p$ *for all* $p \in P$

3. $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \not\models n$ *for all* $n \in N$

$\mathcal{D}$ *is a* minimal diagnosis *iff there is no* $\mathcal{D}' \subset \mathcal{D}$ *such that* $\mathcal{D}'$ *is a diagnosis.* $\mathcal{D}$ *is a* minimum cardinality diagnosis *iff there is no diagnosis* $\mathcal{D}'$ *such that* $|\mathcal{D}'| < |\mathcal{D}|$.[42]

The following proposition of [SFFR12] characterizes diagnoses by replacing EX with the positive test cases.

Corollary 28.1. *Given a DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$*, a set of axioms* $\mathcal{D} \subseteq \mathcal{K}$ *is a diagnosis w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ *iff*

$$(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup \{ \bigwedge_{p \in P} p \}$$

is satisfiable (coherent) and

$$\forall n \in N \ : \ (\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup \{ \bigwedge_{p \in P} p \} \not\models n$$

Hereafter we assume that a diagnosis always exists.

Proposition 28.1. *A diagnosis* $\mathcal{D}$ *w.r.t. a DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ *exists iff* $\mathcal{B} \cup \{ \bigwedge_{p \in P} p \}$ *is consistent (coherent) and* $\forall n \in N \ : \ \mathcal{B} \cup \{ \bigwedge_{p \in P} p \} \not\models n$

For the computation of diagnoses *conflict sets* are usually employed to constrain the search space. A conflict set is the part of the KB that preserves the inconsistency/incoherency.

Definition 28.3. *Given a DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$*, a set of axioms* $CS \subseteq \mathcal{K}$ *is a* conflict set *w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ *iff* $CS \cup \mathcal{B} \cup \{ \bigwedge_{p \in P} p \}$ *is inconsistent (incoherent) or there is an* $n \in N$ *such that* $CS \cup \mathcal{B} \cup \{ \bigwedge_{p \in P} p \} \models n$. CS *is minimal iff there is no* $CS' \subset CS$ *such that* CS' *is a conflict set.*[43]

Minimal conflict sets can be used to compute the set of minimal diagnoses as it is shown in [Rei87]. The idea is that each diagnosis must include at least one element of each minimal conflict set.[44]

Proposition 28.2. $\mathcal{D}$ *is a (minimal) diagnosis w.r.t. the DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ *iff* $\mathcal{D}$ *is a (minimal) hitting set for the set of all minimal conflict sets w.r.t.* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$.

For the generation of a minimal conflict set, diagnosis systems use a divide-and-conquer method (e.g. QUICKXPLAIN [Jun04], for short QX), which we discussed in Sections 4.4.1 and 4.4.2. In the worst case, QX requires $O(|CS| \log(\frac{|\mathcal{K}|}{|CS|}))$ calls to the reasoner, where CS is the returned minimal conflict set.

The computation of minimal diagnoses in KB debugging systems is implemented using Reiter's Hitting Set HS-TREE algorithm [Rei87] (cf. Algorithm 2 in Chapter 4). The algorithm constructs a directed

---

[42]If clear from the context, we will often call $\mathcal{D}$ simply a diagnosis without explicitly stating the DPI w.r.t. which it is a diagnosis in the rest of Part VI.

[43]If clear from the context, we will often call CS simply a conflict set without explicitly stating the DPI w.r.t. which it is a conflict set in the rest of Part VI.

[44]In the rest of Part VI, we consider only minimal conflict sets to avoid the issues concerning the pruning rule [Rei87] described in [GSW89].

tree from the root to the leaves, where each non-leave node is labeled with a minimal conflict set and leave nodes are labeled by ✓ (*no conflicts*) or × (*pruned*).

Each (✓) node corresponds to a minimal diagnosis. The minimality of the diagnoses is guaranteed by the minimality of conflict sets used for labeling the nodes, the pruning rule and the breadth-first strategy of the tree generation. Moreover, because of the breadth-first strategy the minimal diagnoses are generated in increasing order of their cardinality. Under the assumption that diagnoses with lower cardinality are more probable than those with higher cardinality, HS-TREE generates most probable minimal diagnoses first.

Diagnoses Discrimination. For many real-world DPIs, a diagnosis system can return a large number of (minimal) diagnoses. Each minimal diagnosis corresponds to a different set of axioms in the given KB $\mathcal{K}$. All the axioms of any minimal diagnosis might be deleted from $\mathcal{K}$ or changed accordingly in order to formulate a valid $\mathcal{K}^*$. The user may extend the test cases P and N such that diagnoses are eliminated, thus identifying exactly the correct minimal diagnosis. For discriminating between minimal diagnoses we assume that the user knows some of the sentences a valid $\mathcal{K}^*$ must (not) entail, that is the user serves as an oracle.

Property 3. *Given a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$, a set of diagnoses $\mathbf{D}$ w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$, and a logical sentence Q representing the oracle query $\mathcal{K}^* \models Q$. If the oracle gives the answer* yes *then $\mathcal{D}_i \in \mathbf{D}$ is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P \cup \{Q\}, N \rangle$ iff both conditions hold:*

$$(\mathcal{K} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \{ \bigwedge_{p \in P} p \} \cup \{Q\} \text{ is consistent}$$

$$\forall n \in N \; : \; (\mathcal{K} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \{ \bigwedge_{p \in P} p \} \cup \{Q\} \not\models n$$

If the oracle gives the answer no *then $\mathcal{D}_i \in \mathbf{D}$ is a diagnosis w.r.t. $\langle \mathcal{K}, \mathcal{B}, P, N \cup \{Q\} \rangle$ iff both conditions hold:*

$$(\mathcal{K} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \{ \bigwedge_{p \in P} p \} \text{ is consistent}$$

$$\forall n \in (N \cup \{Q\}) \; : \; (\mathcal{K} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \{ \bigwedge_{p \in P} p \} \not\models n$$

However, many different queries might exist for some set of diagnoses $|\mathbf{D}| \geq 2$, in the extreme case exponentially many (in $|\mathbf{D}|$). To select the best query, the authors in [SFFR12] suggest two query selection strategies: SPLIT-IN-HALF (SPL) and ENTROPY (ENT). The first strategy is a greedy approach preferring queries which allow to remove half of the diagnoses in $\mathbf{D}$, for both answers to the query. The second is an information-theoretic measure, which estimates the information gain for both outcomes of each query and returns the query that maximizes the expected information gain. The *prior fault probabilities* required for evaluating the ENT measure can be obtained from statistics of previous diagnosis sessions. For instance, if the user has problems to apply "∃", then the diagnosis logs are likely to contain more repairs of axioms including this quantifier. Consequently, the prior fault probabilities of axioms including "∃" should be higher. Given the fault probabilities of axioms, one can calculate prior fault probabilities of diagnoses as well as evaluate ENT (see [SFFR12] for more details). The queries for both strategies are constructed by exploiting so called classification and realization services provided by description logic reasoners. Given a KB $\mathcal{K}$ and interpreting unary predicates as classes (rsp. concepts), the classification generates the inheritance (subsumption) tree, i.e. the entailments $\mathcal{K} \models \forall X \, p(X) \rightarrow q(X)$, if p is a subclass of q. Realization computes, for each individual name t occurring in a KB $\mathcal{K}$, a set of most specific classes p s.t. $\mathcal{K} \models p(t)$ (see [BCM$^+$07] for details).

Due to the number of diagnoses and the complexity of diagnosis computation, not all diagnoses are exploited for generating queries but a set of minimal diagnoses of size less or equal to some (small) predefined number m [SFFR12]. We call this set the *leading diagnoses* and denote it by $\mathbf{D}$ from now on. This set comprises the (most probable) minimal diagnoses which represent the set of all diagnoses.

The sequential KB debugging process can be sketched as follows. As input a DPI and some meta information, such as prior fault estimates $\mathcal{F}$, query selection strategy s_Q (SPL or ENT) and stop criterion σ, are given. As output a minimal diagnosis is returned that has a posterior probability of at least $1-\sigma$. For sufficiently small σ this means that the returned diagnosis is highly probable whereas all other minimal diagnoses are highly improbable.

1. Using QX and HS-TREE, compute a set of leading diagnoses $\mathbf{D}$ of cardinality $\min(m, a)$, where a is the number of all minimal diagnoses w.r.t. the DPI and m is the number of leading diagnoses predefined by a user.

2. Use the prior fault probabilities $\mathcal{F}$ and the already specified test cases to compute (posterior) probabilities of diagnoses in $\mathbf{D}$ by the Bayesian Rule (cf. [SFFR12]).

3. If some diagnosis $\mathcal{D} \in \mathbf{D}$ has a probability greater than or equal to $1 - \sigma$ or the user accepts $\mathcal{D}$ as the axioms to be changed then stop and return $\mathcal{D}$.

4. Use $\mathbf{D}$ to generate a set of queries and select the best query Q according to s_Q.

5. Ask the user $\mathcal{K}^* \models Q$ and, depending on the answer, add Q either to P or to N.

6. Remove elements from $\mathbf{D}$ violating the newly acquired test case.

7. Repeat at Step 1.

Chapter 29

Interactive Direct Diagnosis of Knowledge Bases

The novelty of our approach is the interactivity combined with the direct calculation of diagnoses. To this end we will utilize an "inverse" version of the QX algorithm [Jun04] called INV-QX and an associated "inverse" version of HS-TREE termed INV-HS-TREE.

This combination of algorithms was first used in [FSZ11]. However, we introduced two modifications: (i) a depth-first search strategy instead of breadth-first and (ii) a new pruning rule which moves axioms from $\mathcal{K}$ to $\mathcal{B}$ instead of just removing them from $\mathcal{K}$, since not adding them to $\mathcal{B}$ might result in losing some of the minimal diagnoses.

INV-QX – Key Idea. INV-QX relies on the monotonic semantics of the used knowledge representation language. The algorithm takes a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ and a ranking heuristic $\prec$ as input and outputs either one minimal diagnosis or 'no diagnosis exists'. The ranking heuristic assigns a fault probability to each axiom in $\mathcal{K}$, if this information is available; otherwise every axiom has the same rank.

The main idea behind Algorithm 15 is to start with the set $\mathcal{D}_0 = \emptyset$ and extend it until a subset of axioms $\mathcal{D} \subseteq \mathcal{K}$ is found such that $\mathcal{D}$ is a minimal diagnosis with respect to Definition 28.2. In the first steps (lines 1-3), Algorithm 15 defines a (potentially) faulty set of axioms $\mathcal{K}'$ and a set $\mathcal{B}'$ of axioms assumed to be correct and sorts $\mathcal{K}'$ w.r.t. the ranking heuristic (SORT). Next, INV-QX verifies whether a diagnosis exists for the input data (line 4), i.e. if the conditions given by Proposition 28.1 are met. This is accomplished by a call to the VERIFY function (defined in line 18 ff.) which requires a reasoner that implements consistency checking (ISCONSISTENT) and allows to decide whether a set of axioms $\mathcal{K}'$ entails some axiom n or not (ENTAILS). Concretely, VERIFY tests for given arguments $\mathcal{B}$ (set of correct axioms), $\mathcal{D}$ (potential minimal diagnosis), $\mathcal{K}$ (potentially faulty set of axioms), N (negative test cases) whether the set $\mathcal{D}$ is a minimal diagnosis or not according to Corollary 28.1. In case no diagnosis exists, the algorithm returns 'no diagnosis exists', otherwise it calls the function FINDDIAG in line 6.

FINDDIAG (line 7) is the main function of the algorithm which takes six arguments as input. The values of the arguments $\mathcal{B}$, $\mathcal{K}$ and N remain constant during the recursion and are required only for the verification of requirements, i.e. calls to the VERIFY function. The values of $\mathcal{D}$ (potential diagnosis), Δ (axioms most recently added to $\mathcal{D}$) and $\mathcal{K}_\Delta$ (part of the original knowledge base that is currently analyzed for the inclusion of axioms that are elements of the sought minimal diagnosis) on the other hand change throughout the recursive calls of FINDDIAG. The two latter sets are obtained by recurrently partitioning the set $\mathcal{K}_\Delta$ (SPLIT and GETELEMENTS in lines 12-14). In most of the implementations SPLIT is specified so as to return $k = \lfloor |\mathcal{K}_\Delta|/2 \rfloor$ which causes the splitting of $\mathcal{K}_\Delta$ into partitions of equal cardinality (this results in the best worst case time complexity [Jun04]). The algorithm pursues this to divide-and-conquer

---

Algorithm 15 INV-QX$(\mathcal{K}, \mathcal{B}, P, N, \prec)$

Input: faulty set of axioms $\mathcal{K}$, set of background axioms $\mathcal{B}$, set of positive test cases P, set of negative test cases N, ranking heuristic $\prec$

Output: a minimal diagnosis $\mathcal{D}$ or 'no diagnosis exists'

1: $\mathcal{K}' \leftarrow \mathcal{K} \setminus \mathcal{B}$
2: $\mathcal{B}' \leftarrow \mathcal{B} \cup P$
3: $\mathcal{K}' \leftarrow \text{SORT}(\mathcal{K}', \prec)$
4: **if** $\neg\text{VERIFY}(\mathcal{B}', \emptyset, \emptyset, N)$ **then**
5:     **return** 'no diagnosis exists'
6: **return** $\text{FINDDIAG}(\mathcal{B}', \emptyset, \mathcal{K}', \mathcal{K}', \mathcal{K}', N)$

7: **procedure** $\text{FINDDIAG}(\mathcal{B}, \mathcal{D}, \Delta, \mathcal{K}_\Delta, \mathcal{K}, N)$ **returns** a minimal diagnosis
8:     **if** $\Delta \neq \emptyset \wedge \text{VERIFY}(\mathcal{B}, \mathcal{D}, \mathcal{K}, N)$ **then**
9:         **return** $\emptyset$
10:     **if** $|\mathcal{K}_\Delta| = 1$ **then**
11:         **return** $\mathcal{K}_\Delta$
12:     $k \leftarrow \text{SPLIT}(|\mathcal{K}_\Delta|)$
13:     $\mathcal{K}_1 \leftarrow \text{GETELEMENTS}(\mathcal{K}_\Delta, 1, k)$
14:     $\mathcal{K}_2 \leftarrow \text{GETELEMENTS}(\mathcal{K}_\Delta, k+1, |\mathcal{K}_\Delta|)$
15:     $\mathcal{D}_2 \leftarrow \text{FINDDIAG}(\mathcal{B}, \mathcal{D} \cup \mathcal{K}_1, \mathcal{K}_1, \mathcal{K}_2, \mathcal{K}, N)$
16:     $\mathcal{D}_1 \leftarrow \text{FINDDIAG}(\mathcal{B}, \mathcal{D} \cup \mathcal{D}_2, \mathcal{D}_2, \mathcal{K}_1, \mathcal{K}, N)$
17:     **return** $\mathcal{D}_1 \cup \mathcal{D}_2$

18: **procedure** $\text{VERIFY}(\mathcal{B}, \mathcal{D}, \mathcal{K}, N)$ **returns** *true* or *false*
19:     $\mathcal{K}' \leftarrow (\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B}$
20:     **if** $\neg\text{ISCONSISTENT}(\mathcal{K}')$ **then**
21:         **return** *false*
22:     **for** $n \in N$ **do**
23:         **if** $\text{ENTAILS}(\mathcal{K}', n)$ **then**
24:             **return** *false*
25:     **return** *true*

---

strategy (lines 15 and 16) until it identifies that the set $\mathcal{D}$ is a diagnosis (line 8). In further iterations the algorithm minimizes this diagnosis by splitting it into sub-diagnoses of the form $\mathcal{D} = \mathcal{D}' \cup \mathcal{K}_\Delta$, where $\mathcal{K}_\Delta$ contains only one axiom. In case $\mathcal{D}$ is a diagnosis and $\mathcal{D}'$ is not, the algorithm decides that $\mathcal{K}_\Delta$ is a subset of the sought minimal diagnosis. Just as the original QX algorithm, INV-QX always terminates and it returns a minimal diagnosis for a given DPI (provided there exists one).

INV-QX requires $O(|\mathcal{D}| \log(\frac{|\mathcal{K}|}{|\mathcal{D}|}))$ calls to a reasoner to find a minimal diagnosis $\mathcal{D}$. Moreover, in opposite to SAT or CSP methods, e.g. [NPQW13], INV-QX can be used to compute diagnoses in cases when satisfiability checking is beyond NP. For instance, reasoning for most of the KBs used in Chapter 30 is EXPTIME-complete.

INV-QX is a deterministic algorithm and returns one and the same minimal diagnosis if applied twice to one and the same DPI. In order to obtain a different next diagnosis, the DPI used as input for INV-QX must be modified accordingly. To this end, we employ the INV-HS-TREE algorithm.

INV-HS-TREE – Construction. The algorithm is inverse to the HS-TREE algorithm in the sense that nodes are now labeled by minimal diagnoses (instead of minimal conflict sets) and a path from the root to an open node is a partial conflict set (instead of a partial diagnosis). The algorithm constructs a directed tree from the root to the leaves, where each node nd is labeled either with a minimal diagnosis

$\mathcal{D}$ or $\times$ (*pruned*) which indicates that the node is closed. For each $s \in \mathcal{D}$ there is an outgoing edge labeled by s. Let $H(\mathsf{nd})$ be the set of edge labels on the path from the root to the node nd. Initially the algorithm generates an empty root node and adds it to a LIFO-queue, thereby implementing a *depth-first search* strategy. Until the required number m of minimal diagnoses is reached or the queue is empty, the algorithm removes the first node nd from the queue and labels nd by applying the following steps:

1. (*reuse*): $\mathcal{D} \in \mathbf{D}$ if $\mathcal{D} \cap H(\mathsf{nd}) = \emptyset$, add for each $s \in \mathcal{D}$ a node to the LIFO-queue, or

2. (*pruned*): $\times$ if Inv-QX($\mathcal{K} \setminus H(\mathsf{nd}), \mathcal{B} \cup H(\mathsf{nd}), P, N$) = 'no-diagnosis-exists', (according to Proposition 28.1), or

3. (*compute*): $\mathcal{D}$ if Inv-QX($\mathcal{K} \setminus H(\mathsf{nd}), \mathcal{B} \cup H(\mathsf{nd}), P, N$) = $\mathcal{D}$; add $\mathcal{D}$ to $\mathbf{D}$ and add for each $s \in \mathcal{D}$ a node to the LIFO-queue.

Reuse of known diagnoses in Step 1 and the addition of $H(\mathsf{nd})$ to the background theory $\mathcal{B}$ in Steps 2 and 3 allows the algorithm to force Inv-QX to search for a minimal diagnosis that is different to all already computed minimal diagnoses in $\mathbf{D}$. So, if neither Step 1 nor Step 2 are applicable, Inv-HS-Tree calls Inv-QX which is guaranteed to compute a *new* minimal diagnosis $\mathcal{D}$ which is then added to the set $\mathbf{D}$.

Inv-HS-Tree – Update Procedure for Interactivity. Since paths in Inv-HS-Tree are (1) irrelevant and need not be maintained, and (2) only a small (linear) number of nodes/paths is in memory due to the application of a depth-first search, the update procedure after a query Q has been answered involves a reconstruction of the tree. In particular, by answering Q, $m - k$ of (maximally) m leading diagnoses are invalidated and deleted from memory. The k still valid minimal diagnoses are used to build a new tree. To this end, the root is labeled by any of these k minimal diagnoses and a tree is constructed as described above where the k diagnoses are incorporated for the *reuse* check. Note that the recalculation of a diagnosis that has been invalidated by a query is impossible as in subsequent iterations a new DPI is considered which includes the answered query as a test case.

Inv-HS-Tree – Comparison to HS-Tree. Since Inv-QX($\mathcal{K}, \mathcal{B} \cup H(\mathsf{nd}), P, N$) = 'no diagnosis exists' means $H(\mathsf{nd})$ is a conflict set w.r.t. the current DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$, in Inv-HS-Tree any path that is a conflict set is *automatically* closed. This makes a pruning rule similar to the one in HS-Tree which closes a node nd given an alternative path $H(\mathsf{nd}')$ to a closed node nd' with $H(\mathsf{nd}') \subseteq H(\mathsf{nd})$ obsolete. So, Inv-HS-Tree benefits from the fact that minimality of diagnoses is independent of path-minimality, and thereby might save time for comparison of exponentially many paths over HS-Tree.

Another great advantage of Inv-HS-Tree over HS-Tree is that it can be constructed using a space-saving depth-first strategy. The reason for this is again that minimality of paths (conflict sets) is irrelevant in Inv-HS-Tree whereas in HS-Tree minimality of paths (diagnoses) is essential. In an implementation where successors of a node are generated one at a time in Inv-HS-Tree, the space complexity of the entire tree construction is *linear* and amounts to $O(2m) = O(m)$ where m is the predefined maximum number of leading diagnoses. This holds as $k < m$ still valid diagnoses from the previous iteration are in memory, plus a path in the tree can comprise a maximum of m nodes corresponding to *different* (reused or new) diagnoses before the search is stopped ($|\mathbf{D}| = m$). No conflict sets are stored.

For HS-Tree, by contrast, the worst-case space complexity is *exponential*, i.e. $O(|CS_{\max}|^d)$ where $|CS_{\max}|$ is the size of the minimal conflict set with maximum cardinality (among all minimal conflict sets w.r.t. the given DPI) and d is the tree depth were m minimal diagnoses have been generated.

The crucial disadvantage of Inv-HS-Tree compared to HS-Tree is that the former cannot guarantee the computation of diagnoses in a special order, e.g. minimum cardinality or maximum fault probability first.
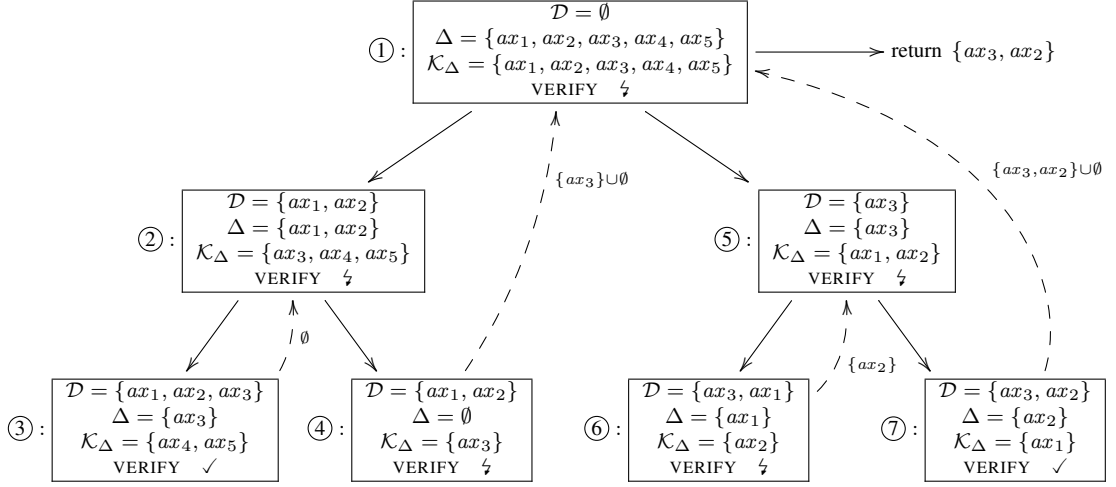
Figure 29.1: INV-QX recursion tree. Each node shows values of FINDDIAG input variables as well as the result of the VERIFY function called in line 8.
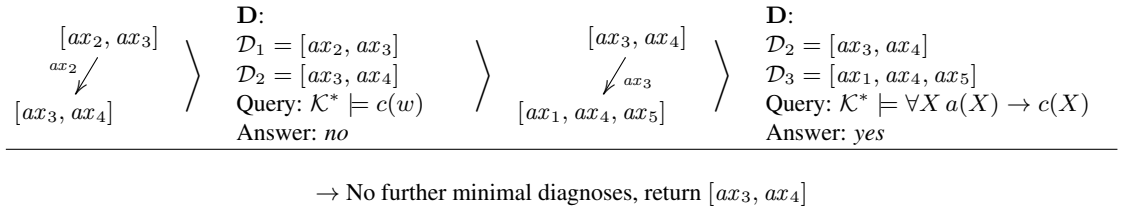

$\rightarrow$ No further minimal diagnoses, return $[ax_3, ax_4]$

Figure 29.2: Identification of the target diagnosis $[ax_3, ax_4]$ using INV-HS-TREE.

Example 29.1   Consider a DPI with the following knowledge base $\mathcal{K}$:

$$ax_1 : \forall X\, c(X) \rightarrow a(X) \qquad ax_4 : \forall X\, b(X) \rightarrow c(X)$$
$$ax_2 : \forall X\, c(X) \rightarrow e(X) \qquad ax_5 : \forall X\, b(X) \rightarrow \neg d(X)$$
$$ax_3 : \forall X\, a(X) \rightarrow \neg(c(X) \vee \neg b(X))$$

the background knowledge $\mathcal{B} = \{a(v), b(w), c(s)\}$, one positive $P = \{d(v)\}$ and one negative $N = \{e(w)\}$ test case.

Let us first show how a minimal diagnosis is computed by INV-QX (see Figure 29.1). The algorithm starts with an empty diagnosis $\mathcal{D} = \emptyset$ and $\mathcal{K}_\Delta$ containing all axioms of $\mathcal{K}$ ①. VERIFY called in line 8 returns *false* since $(\mathcal{B} \cup P) \cup (\mathcal{K} \setminus \emptyset)$ is inconsistent. Since moreover $|\mathcal{K}_\Delta| \neq 1$ (line 10), the algorithm splits $\mathcal{K}_\Delta$ into $\{ax_1, ax_2\}$ and $\{ax_3, ax_4, ax_5\}$ (lines 12-14) and passes the sub-problem (line 15) to the next level of recursion ②. Since the set $\mathcal{D} = \{ax_1, ax_2\}$ is not a diagnosis, i.e. the KB $(\mathcal{B} \cup P) \cup (\mathcal{K} \setminus \mathcal{D})$ is inconsistent and $|\mathcal{K}_\Delta| = |\{ax_3, ax_4, ax_5\}| \neq 1$, the problem in $\mathcal{K}_\Delta$ is split one more time (lines 12-14). On the second level of recursion ③ the set $\mathcal{D}$ is a diagnosis, yet not a minimal one. The function VERIFY returns *true* and the algorithm starts to analyze the found diagnosis. Therefore, it verifies whether the last extension of the set $\mathcal{D}$ is a subset of a minimal diagnosis ④. Since the extension includes only one axiom ax_3 and the extended set $\{ax_1, ax_2\}$ is not a diagnosis, the algorithm concludes that ax_3 must be an element of the a minimal diagnosis. The leftmost branch of the recursion tree terminates and returns

$$\left\langle \begin{array}{c} \langle ax_1, ax_3 \rangle^C \\ \swarrow_{ax_1} \quad \searrow^{ax_3} \\ \langle ax_2, ax_4 \rangle^C \quad \langle ax_2, ax_4 \rangle^R \\ _{ax_2}\swarrow \quad \searrow_{ax_4} \quad _{ax_2}\downarrow \quad \searrow_{ax_4} \\ \langle ax_3, ax_5 \rangle^C \quad \langle ax_3, ax_5 \rangle^R \; \checkmark \quad \checkmark \end{array} \right\rangle$$

Minimal diagnoses:
$\mathcal{D}_1 = [ax_3, ax_4]$
$\mathcal{D}_2 = [ax_2, ax_3]$

Query: $\mathcal{K}^* \models c(w)$
Answer: *no*

$$\Big\rangle$$

Iteration 1

$$\left\langle \begin{array}{c} \langle ax_1, ax_3 \rangle^C \\ _{ax_1}\swarrow \quad \searrow^{ax_3} \\ \langle ax_2, ax_4 \rangle^C \quad \langle ax_2, ax_4 \rangle^R \\ _{ax_2}\swarrow \quad \searrow_{ax_4} \quad \downarrow_{ax_2} \quad \searrow^{ax_4} \\ \langle ax_3, ax_5 \rangle^C \quad \langle ax_3, ax_5 \rangle^R \quad \times \quad \checkmark \\ \swarrow_{ax_3} \quad \searrow_{ax_5} \quad \downarrow_{ax_3} \quad \searrow^{ax_5} \\ \times \quad \langle ax_3, ax_4 \rangle^C \quad \times \quad \checkmark \end{array} \right\rangle$$

Minimal diagnoses:
$\mathcal{D}_1 = [ax_3, ax_4]$
$\mathcal{D}_3 = [ax_1, ax_4, ax_5]$

Query: $\mathcal{K}^* \models \forall X\ a(X) \to c(X)$
Answer: *yes*

No further minimal
diagnoses, return
$\mathbf{D} = \{[ax_4, ax_3]\}$
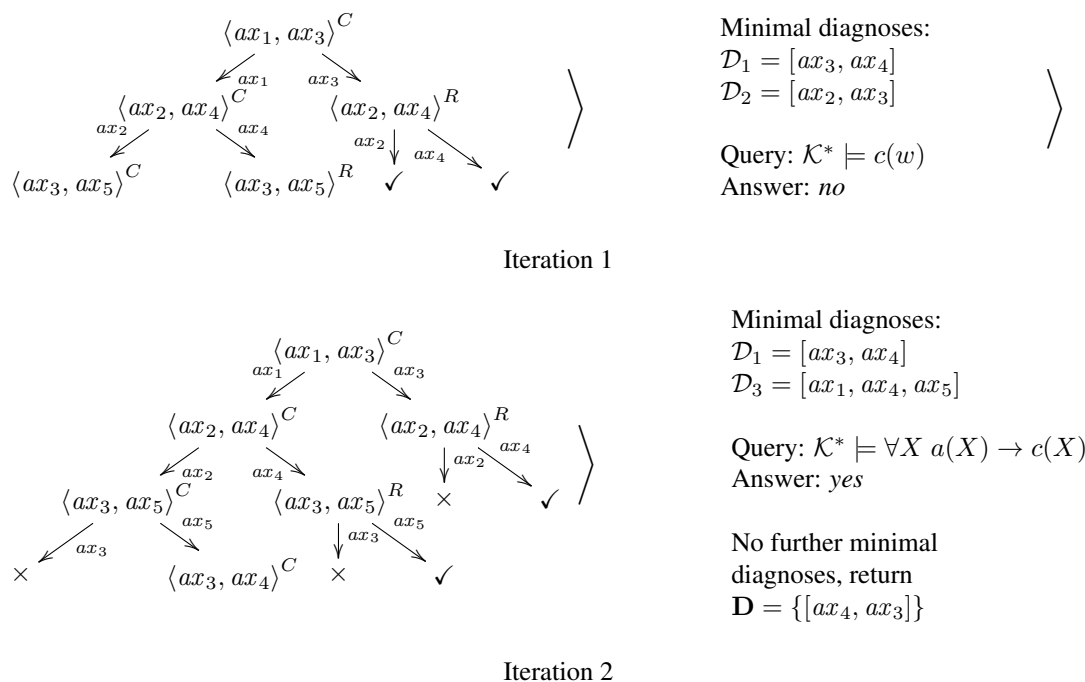
Iteration 2

Figure 29.3: Identification of the target diagnosis $[ax_4, ax_3]$ using HS-TREE and QX computing conflicts on-demand. All computed node labels are denoted with C and all reused with R.

$\{ax_3\}$. This axiom is added to the set $\mathcal{D}$ and the algorithm starts investigating whether the two axioms $\{ax_1, ax_2\}$ also belong to a minimal diagnosis ⑤. First, it tests the set $\{ax_3, ax_1\}$ ⑥, which is not a diagnosis, and in the next iteration it identifies $\{ax_3, ax_2\}$ as a minimal diagnosis in node ⑦ which is the final output of INV-QX.

In general, for the sample DPI there are three minimal diagnoses $\{\mathcal{D}_1 : [ax_2, ax_3], \mathcal{D}_2 : [ax_3, ax_4], \mathcal{D}_3 : [ax_1, ax_4, ax_5]\}$ and four minimal conflict sets $\{CS_1 : \langle ax_1, ax_3 \rangle, CS_2 : \langle ax_2, ax_4 \rangle, CS_3 : \langle ax_3, ax_5 \rangle, CS_4 : \langle ax_3, ax_4 \rangle\}$.

Now we show how INV-HS-TREE can be applied to find the (correct) diagnosis that allows the formulation of a valid KB (with the desired semantics in terms of entailments and non-entailments). Assume that the number of leading diagnoses required for query generation is set to $m = 2$. Applied to the sample DPI, INV-HS-TREE computes a minimal diagnosis $\mathcal{D}_1 := [ax_2, ax_3] = \text{INV-QX}(\mathcal{K}, \mathcal{B}, P, N)$ to label the root node, see Figure 29.2. Next, it generates one successor node that is linked with the root by an edge labeled with ax_2. For this node $\text{INV-QX}(\mathcal{K} \setminus \{ax_2\}, \mathcal{B} \cup \{ax_2\}, P, N)$ yields a minimal diagnosis $\mathcal{D}_2 := [ax_3, ax_4]$ disjoint with $\{ax_2\}$. Now $|\mathbf{D}| = 2$ and a query is generated and answered as in Figure 29.2. Adding $c(w)$ to the negative test cases invalidates $\mathcal{D}_1$ since $(\mathcal{K} \setminus \mathcal{D}_1) \cup \mathcal{B} \cup P \models c(w)$. In the course of the update, $\mathcal{D}_1$ is deleted and $\mathcal{D}_2$ used as the root of a new tree. An edge labeled with ax_3 is created and diagnosis $\mathcal{D}_3 := [ax_1, ax_4, ax_5]$ is generated. After the answer to the second query is added to the positive test cases, $\mathcal{D}_3$ is invalidated and all outgoing edge labels ax_3, ax_4 of the root $\mathcal{D}_2$ of the new tree are conflict sets for the current DPI $\langle \mathcal{K}, \mathcal{B}, \{d(v), \forall X\ a(X) \to c(X)\}, \{e(w), c(w)\}\rangle$, i.e. all leaf nodes are labeled by $\times$ and the tree construction is complete. So, $\mathcal{D}_2$ is returned as its probability is 1.

Finally, let us compare the performance of HS-TREE [Rei87] with the one of INV-HS-TREE. Applied to our sample DPI, the standard interactive diagnosis process using HS-TREE first calls QX [Jun04]

which returns a minimal conflict set $\langle ax_1, ax_3 \rangle$ (Figure 29.3). This minimal conflict set is used to label the root node of the HS-TREE. By reuse (R) of already computed minimal conflict sets or further calls (C) to QX (if there is no conflict set to reuse) the algorithm extends the HS-TREE until $m = 2$ leading minimal diagnoses $\mathbf{D} := \{\mathcal{D}_1, \mathcal{D}_2\}$ for the DPI are computed. To discriminate between diagnoses in $\mathbf{D}$, the query $\mathcal{K}^* \models c(w)$ is computed. Given the answer *no*, $\mathcal{D}_2$ is invalidated which is reflected by the closing of the corresponding node in the tree (label $\times$). The second iteration considers the new DPI $\langle \mathcal{K}, \mathcal{B}, \{d(v)\}, \{e(w), c(w)\} \rangle$ and involves further expansion of (open nodes in) the tree under consideration of the pruning rule until the size of leading diagnoses $\mathbf{D}$ is 2, i.e. $\{\mathcal{D}_1, \mathcal{D}_3\}$. After the positive answer to the second query and closing of the invalidated diagnosis $\mathcal{D}_3$, the recalculation of $\mathbf{D}$ (not shown in Figure 29.3) yields no further minimal diagnoses. So, the algorithm terminates and returns $\mathcal{D}_1$. As we can see, HS-TREE comprises a lot of intermediate nodes in comparison to INV-HS-TREE. That leads to a dramatic difference in memory consumption between these two approaches. $\qquad\square$

Chapter 30

Evaluation

We evaluated our approach DIR (based on INV-QX and INV-HS-TREE) versus the standard technique STD [SFFR12] (based on QX and HS-TREE) using a set of KBs created by automatic matching systems. Given two knowledge bases $\mathcal{K}_i$ and $\mathcal{K}_j$, a matching system outputs an *alignment* M_{ij} which is a set of *correspondences* between semantically related entities of $\mathcal{K}_i$ and $\mathcal{K}_j$. Let $Q(\mathcal{K})$ denote the set of all elements of $\mathcal{K}$ for which correspondences can be produced, i.e. names of predicates. Each correspondence is a tuple $\langle x_i, x_j, r, v \rangle$, where $x_i \in Q(\mathcal{K}_i)$, $x_j \in Q(\mathcal{K}_j)$ and x_i, x_j have the same arity, $r \in \{\leftarrow, \leftrightarrow, \rightarrow\}$ is a logical operator and $v \in [0,1]$ is a confidence value. The latter expresses the probability of a correspondence to be correct. Let $\overline{X}$ be a vector of distinct logical variables with a length equal to the arity of x_i, then each $\langle x_i, x_j, r, v \rangle \in M_{ij}$ is translated to the axiom $\forall \overline{X}\, x_i(\overline{X})\; r\; x_j(\overline{X})$. Let $\mathcal{K}(M_{ij})$ denote the set of axioms resulting from such a translation for the alignment M_{ij}. Then the result of the matching process is an aligned KB $\mathcal{K}_{ij} = \mathcal{K}_i \cup \mathcal{K}(M_{ij}) \cup \mathcal{K}_j$.

The KBs considered in this section were created by ontology matching systems participating in the Ontology Alignment Evaluation Initiative (OAEI) 2011 [EFvH$^+$11]. Each matching experiment in the framework of OAEI represents a scenario in which a user obtains an alignment M_{ij} by means of some (semi)automatic tool for two real-world *ontologies* $\mathcal{K}_i$ and $\mathcal{K}_j$. The latter are KBs expressed by the Web Ontology Language (OWL) [GHM$^+$08] whose semantics is compatible with the $\mathcal{SROIQ}$ description logic (DL). This DL is a decidable fragment of first-order logic for which a number of effective reasoning methods exist [BCM$^+$07]. Note that, $\mathcal{SROIQ}$ is a member of a broad family of DL knowledge representation languages. All DL KBs considered in this evaluation are expressible in $\mathcal{SROIQ}$.

The goal of the first experiment was to compare the performance of STD and DIR on a set of large, but diagnostically uncomplicated KBs, generated for the Anatomy experiment of OAEI.[45] In this experiment the matching systems had to find correspondences between two KBs describing the human and the mouse anatomy. $\mathcal{K}_1$ (Human) and $\mathcal{K}_2$ (Mouse) include 11545 and 4838 axioms, respectively, whereas the size of the alignment M_{12} produced by different matchers varies between 1147 and 1461 correspondences. Seven matching systems produced a classifiable but incoherent output. One system generated a classifiable and coherent aligned KB. However, this system employs a built-in heuristic diagnosis engine which does not guarantee to produce minimal diagnoses. That is, some axioms are removed without reason. Four systems produced KBs which could not be processed by current reasoning systems (e.g. HermiT) since these KBs could not be classified within 2 hours.

For testing the performance of our system we have to define the correct output of sequential diagnosis which we call the target diagnosis $\mathcal{D}_t$. We assume that the only available knowledge is M_{ij} together with $\mathcal{K}_i$ and $\mathcal{K}_j$. In order to measure the performance of the matching systems the organizers of OAEI

---

[45] All KBs and source code of programs used in the evaluation can be downloaded from http://code.google.com/p/rmbd/wiki/DirectDiagnosis. The tests were performed on Core i7, 64GB RAM running Ubuntu, Java 7 and HermiT as DL reasoner.

System	Scoring	HS-Tree			Inv-HS-Tree		
		Time	#Queries	Reaction	Time	#Queries	Reaction
AgrMaker	ENT	19.62	1	19.10	20.83	1	18.23
AgrMaker	SPL	36.04	4	8.76	36.03	4	8.28
GOMMA-bk	ENT	18.34	1	18.07	14.47	1	12.68
GOMMA-bk	SPL	18.95	3	6.15	19.51	3	5.91
GOMMA-nobk	ENT	18.26	1	17.98	14.26	1	12.49
GOMMA-nobk	SPL	18.74	3	6.08	19.47	3	5.89
Lily	ENT	78.54	1	77.71	82.52	1	72.83
Lily	SPL	82.94	4	20.23	115.24	4	26.93
LogMap	ENT	6.60	1	6.30	13.41	1	11.36
LogMap	SPL	6.61	2	3.17	15.13	2	6.82
LogMapLt	ENT	14.85	1	14.54	12.89	1	11.34
LogMapLt	SPL	15.59	3	5.05	17.45	3	5.29
MapSSS	ENT	81.06	4	19.86	56.17	3	17.32
MapSSS	SPL	88.32	5	17.26	77.59	6	12.43

Table 30.1: HS-Tree and Inv-HS-Tree applied to Anatomy benchmark. Time is given in sec, **Scoring** stands for query selection strategy, **Reaction** is the average system reaction time between queries.

provided a *golden standard* alignment M_t considered as correct. Nevertheless, we cannot assume that M_t is explicitly available since the matching system would have used this information. W.r.t. the knowledge available, any minimal diagnosis w.r.t. the DPI $\langle \mathcal{K}(M_{ij}), \mathcal{K}_i \cup \mathcal{K}_j, \emptyset, \emptyset \rangle$ (i.e. $\mathcal{K}(M_{ij})$ is the KB and $\mathcal{K}_i \cup \mathcal{K}_j$ used as background theory) can be selected as $\mathcal{D}_t$. However, for every alignment we selected a minimal diagnosis as target diagnosis $\mathcal{D}_t$ which is outside the golden standard. By this procedure we mimic cases where additional information can be acquired such that no correspondence of the golden standard is removed in order to establish coherence. We stress that this setting is unfavorable for diagnosis since providing more information by exploiting the golden standard would reduce the number of queries to ask. Consequently, we limit the knowledge to $\mathcal{K}_{ij}$ and use $\mathcal{K}_{ij} \setminus \mathcal{D}_t$ to answer the queries.

In particular, the selection of a target diagnosis $\mathcal{D}_t$ for each $\mathcal{K}_{ij}$ output by a matching system was done in two steps: (i) compute the set of all minimal diagnoses **AD** w.r.t. the correspondences which are not in the golden standard, i.e. $\mathcal{K}(M_{ij} \setminus M_t)$, and use $\mathcal{K}_i \cup \mathcal{K}_j \cup \mathcal{K}(M_{ij} \cap M_t)$ as background theory. The set of test cases are empty. I.e. the DPI is $\langle \mathcal{K}(M_{ij} \setminus M_t), \mathcal{K}_i \cup \mathcal{K}_j \cup \mathcal{K}(M_{ij} \cap M_t), \emptyset, \emptyset \rangle$. (ii) select $\mathcal{D}_t$ randomly from **AD**. The prior fault probabilities of axioms $ax \in \mathcal{K}(M_{ij})$ expressing correspondences were set to $1 - v_{ax}$ where v_{ax} is the confidence value provided by the matcher.

The tests were performed for the mentioned seven incoherent alignments where the input DPI is $\langle \mathcal{K}(M_{ij}), \mathcal{K}_i \cup \mathcal{K}_j, \emptyset, \emptyset \rangle$ and the output is a minimal diagnosis. We tested DIR and STD with both query selection strategies SPLIT-IN-HALF (SPL) and ENTROPY (ENT) in order to evaluate the quality of fault probabilities based on confidence values. Moreover, for generating a query, the number of leading diagnoses was limited to $m = 9$.

The results of the first experiment are presented in Table 30.1. DIR computed $\mathcal{D}_t$ within 36 sec. on average and slightly outperformed STD which required 36.7 sec. The number of asked queries was equal for both methods in all but two cases resulting from KBs produced by the `MapSSS` system. For these KBs, DIR required one query more using ENT and one query less using SPL. In general, the results obtained for the Anatomy case show that DIR and STD have similar performance in both runtime and number of queries. Both DIR and STD identified the target diagnosis. Moreover, the confidence values

Ontology (Expressivity)	30 Diag	min |**D**|		Scoring	Time	#Queries	Reaction	#CC	CC
ldoa-conference-confof	48.06	16		ENT	11.6	6	1.5	430	0.003
$\mathcal{SHIN(D)}$				SPL	11.3	7	1.6	365	0.004
ldoa-cmt-ekaw	42.28	12		ENT	48.6	21	2.2	603	0.016
$\mathcal{SHIN(D)}$				SPL	139.1	49	2.8	609	0.054
mappso-confof-ekaw	55.66	10		ENT	10	5	1.9	341	0.007
$\mathcal{SHIN(D)}$				SPL	31.6	13	2.3	392	0.021
optima-conference-ekaw	62.13	19		ENT	16.8	5	2.6	553	0.008
$\mathcal{SHIN(D)}$				SPL	16.1	8	1.9	343	0.012
optima-confof-ekaw	44.52	16		ENT	24	20	1.1	313	0.014
$\mathcal{SHIN(D)}$				SPL	17.6	10	1.7	501	0.006
ldoa-conference-ekaw	56.98	16		ENT	56.7	35	1.5	253	0.053
$\mathcal{SHIN(D)}$				SPL	25.5	9	2.7	411	0.016
csa-conference-ekaw	62.82	17		ENT	6.7	2	2.8	499	0.003
$\mathcal{SHIN(D)}$				SPL	22.7	8	2.7	345	0.02
mappso-conference-ekaw	70.46	19		ENT	27.5	13	1.9	274	0.028
$\mathcal{SHIN(D)}$				SPL	71	16	4.2	519	0.041
ldoa-cmt-edas	15.47	16		ENT	24.7	22	1	303	0.008
$\mathcal{ALCOIN(D)}$				SPL	11.2	7	1.4	455	0.002
csa-conference-edas	39.74	26		ENT	18.4	6	2.7	419	0.005
$\mathcal{ALCHOIN(D)}$				SPL	240.8	37	6.3	859	0.036
csa-edas-iasted	377.36	20		ENT	1744.6	3	349.2	1021	1.3
$\mathcal{ALCOIN(D)}$				SPL	7751.9	8	795.5	577	11.5
ldoa-ekaw-iasted	229.72	13		ENT	23871.5	9	1886	287	72.6
$\mathcal{SHIN(D)}$				SPL	20449	9	2100.1	517	37.2
mappso-edas-iasted	293.74	27		ENT	18400.3	5	2028.3	723	17.8
$\mathcal{ALCOIN(D)}$				SPL	159299	11	13116.6	698	213.2

Table 30.2: Sequential diagnosis using direct computation of diagnoses. **30 Diag** is the time required to find 30 minimal diagnoses, **min** $|\mathcal{D}|$ is the cardinality of a minimum cardinality diagnosis, **Scoring** indicates the query selection strategy, **Reaction** is the average system reaction time between queries, **#CC** number of consistency checks, **CC** gives average time needed for one consistency check. Time is given in sec.

provided by the matching systems appeared to be a good estimate for fault probabilities. Thus, in many cases ENT was able to find $\mathcal{D}_t$ using one query only, whereas SPL used 4 queries on average.

In the first experiment, the identification of the target diagnosis by sequential STD required the computation of 19 minimal conflicts on average. Moreover, the average size of a minimum cardinality diagnosis over all KBs in this experiment was 7. In the second experiment (see below), where STD is not applicable, the cardinality of the target diagnosis is significantly higher.

The second experiment was performed on KBs of the OAEI Conference benchmark which turned out to be problematic for STD. For these KBs we observed that the minimum cardinality diagnoses comprise 18 elements on average. In 11 of the 13 KBs of the second experiment (see Table 30.2), STD was unable to find any diagnosis within 2 hours. In the other two cases STD succeeded to find one minimal diagnosis for `csa-conference-ekaw` and nine for `ldoa-conference-confof`. However, DIR even succeeded to find 30 minimal diagnoses for each KB within time acceptable for interactive diagnosis settings. Moreover, on average DIR was able to find 1 minimal diagnosis in 8.9 sec., 9 minimal diagnoses in 40.83 sec. and 30 minimal diagnoses in 107.61 sec. (see Column 2 of Table 30.2). This result shows that DIR is a stable and practically applicable method even in cases where a knowledge base comprises high-cardinality faults.

In the Conference experiment, we first selected the target diagnosis $\mathcal{D}_t$ for each $\mathcal{K}_{ij}$ just as it was done in the described Anatomy case. Next, we evaluated the performance of sequential DIR using both query selection methods. The results of the experiment presented in Table 30.2 show that DIR found $\mathcal{D}_t$ for each KB. On average DIR solved the problems more efficiently using ENT than SPL because also in the Conference case the confidence values provided a reasonable estimation of axiom fault probabilities. Only in three cases ENT required more queries than SPL.

Moreover, the experiments show that the efficiency of debugging methods depends highly on the runtime of the underlying reasoner. For instance, in the hardest case consistency checking took 93.4% of the total time whereas all other operations – including construction of the search tree, generation and selection of queries – took only 6.6% of time. Consequently, sequential DIR requires only a small fraction of computation effort. Runtime improvements can be achieved by advances in reasoning algorithms or the reduction of the number of consistency checks. Currently, in order to generate a query, DIR requires $O(m * |\mathcal{D}| \log(\frac{|\mathcal{K}|}{|\mathcal{D}|}))$ checks to find m leading diagnoses.

A further source for improvements can be observed for the `ldoa-ekaw-iasted` ontology where both methods asked the same number of queries. In this case, a sequential diagnosis session using ENT query selection method required only half of the consistency checks SPL did. However, an average consistency check made in the session using ENT took almost twice as long as an average consistency check using SPL. The analysis of this ontology showed that there is a small subset of axioms (called "hot spot" in [GPS12]) which made reasoning considerably harder. As practice shows, they can be resolved by suitable queries. This can be observed in the `ldoa-ekaw-iasted` case where SPL acquired appropriate test cases early and thereby found $\mathcal{D}_t$ faster. Therefore, research and application of methods allowing fast identification of such hot spots might result in a significant improvement of diagnosis runtime.

Chapter 31

Summary and Conclusions

In this part, we presented a sequential diagnosis method for faulty KBs which is based on the direct computation of minimal diagnoses. We were able to reduce the number of consistency checks by avoiding the computation of minimized conflict sets and by computing just *some* set of minimal diagnoses instead of a set of most probable diagnoses or a set of minimum cardinality diagnoses. The presented evaluation results in Chapter 30 indicate that the performance of the suggested sequential diagnosis system is either comparable with or outperforms the existing approach in terms of runtime and required number of queries in case a KB includes a large number of faults. The scalability of the algorithms was demonstrated on a set of large KBs including thousands of axioms.

Part VII

Epilog

In this part we provide a discussion of related work in Chapter 32,[46] summarize the contributions of this work in Chapter 33 and deal with our future work topics in Chapter 34.

---

[46]Note that related work specific to topics addressed in Parts IV-VI is separately treated in these parts.

Chapter 32

Related Work

To the best of our knowledge no interactive KB debugging methods that ask a user automatically selected queries have been proposed to repair faulty (monotonic) KBs so far (except for our own previous works [SF10, SFFR12, RSFF13, SFRF14c]).

Non-interactive debugging methods for KBs (ontologies) are introduced in [SHCH07, KPHS07, FS05]. Ranking of diagnoses and proposing a "best" diagnosis is presented in [KPSCG06]. This method uses a number of measures such as (a) the frequency with which a formula appears in conflict sets, (b) the impact on the KB in terms of its "lost" entailments when some formula is modified or removed, (c) provenance information about the formula and (d) syntactic relevance of a formula. All these measures are evaluated for each formula in a conflict set. The scores are then combined in a rank value which is associated with the corresponding formula. These ranks are then used by a modified hitting set tree algorithm that identifies diagnoses with a minimal rank. In this work no query generation and selection strategy is proposed if the intended diagnosis cannot be determined reliably with the given a-priori knowledge. In our work additional information is acquired until the minimal diagnosis with the intended semantics can be identified with confidence. In general, the work of [KPSCG06] can be combined with the approaches presented in our work as ranks of logical formulas can be taken into account together with other observations for calculating the prior probabilities of minimal diagnoses (see Section 4.6.1).

The idea of selecting the next query based on certain query selection measures was exploited in the generation of decisions trees [Qui86] and for selecting measurements in the model-based diagnosis of circuits [dKW87] (in both works, the minimal expected entropy measure was used). We extended these methods to query selection in the domain of KB debugging [SF10] and devised further query selection measures [SFFR12, RSFF13].

An approach for the debugging of faulty aligned KBs (ontologies) was proposed by [Mei11]. An aligned KB is the union of two KBs $\mathcal{K}_1$ and $\mathcal{K}_2$ and an alignment $A_{1,2}$ (which is properly formatted as a set of logical formulas, cf. Definition 18 in [Mei11]). $A_{1,2}$ is a set of correspondences (each with an associated automatically computed confidence value) produced by an automatic system (an ontology matcher) given $\mathcal{K}_1$ and $\mathcal{K}_2$ as inputs where each correspondence represents a (possible) semantic relationship between a term occurring in the first and a term occurring in the second input KB. The goal of a debugging system for faulty aligned KBs is usually the determination of a subset of the alignment $A'_{1,2} \subset A_{1,2}$ such that the aligned KB using $A'_{1,2}$ is not faulty. In terms of our approaches, this corresponds to the setting $\mathcal{K} := A_{1,2}$ and $\mathcal{B} := \mathcal{K}_1 \cup \mathcal{K}_2$. We have already shown in [RSFF12, SFRF12] that our systems can also be applied for fault localization in aligned KBs. The work of [Mei11] describes approximate algorithms for computing a "local optimal diagnosis" and complete methods to discover a "global optimal diagnosis". Optimality in this context refers to the maximum sum of confidences in the resulting repaired alignment $A'_{1,2}$. In contrast to our framework, diagnoses are determined automatically

without support for user interaction. Instead, [Mei11] demonstrates techniques for the manual revision of the alignment as a procedure *independent* from debugging. Another difference to our approach is the way of detecting sources of faults. We rely on a divide-and-conquer algorithm [Jun04] for the identification of a minimal conflict set $C \subseteq A_{1,2}$ (in [Mei11] C is called a MIPS, cf. [FS05, SHCH07]). In the worst case the method we use exhibits only $O(|C| * \log(|A_{1,2}|/|C|))$ calls of some function that performs a check for faults in a KB and internally uses a reasoner (in our case ISKBVALID, see Algorithm 1). The "shrink" strategy applied in [Mei11] (which is similar to the "expand-and-shrink" method used in [KPHS07]), on the other hand, requires a worst case number of $O(|A_{1,2}|)$ calls to such a function. Empirical evaluations and a theoretical analysis of the best and worst case complexity of the "expand-and-shrink" method compared to the divide-and-conquer method performed in [SFJ08] revealed that the latter is preferable over the former. It should be noted that a similar divide-and-conquer method as used in our work could most probably also be plugged into the system in [Mei11] instead of the "shrink" method.

There are some ontology matchers which incorporate alignment repair features: CODI [HSNM11], YAM++ [NB12], ASMOV [JMSK09] and KOSIMap [RP10], for instance, employ logic-based techniques to search for a set of predefined "anti-patterns" which must not occur in the aligned ontology, either to avoid inconsistencies or incoherencies or to eliminate unwanted or redundant entailments. In case such a pattern is revealed, it is resolved by eliminating from the alignment some correspondences responsible for its occurrence. All the techniques incorporated in these matchers are distinct from the presented approaches in that they implement incomplete or approximate methods of alignment repair, i.e. not all alternative solutions to the alignment debugging problem are taken into account. As a consequence of this, on the one hand, the final alignment produced by these systems may still trigger faults in the aligned KB. On the other hand, a suboptimal solution may be found, e.g. in terms of the user-intended semantics w.r.t. the aligned ontology or other criteria such as alignment confidence or cardinality.

Another ontology matcher, LogMap 2 [JRGZH12], provides integrated debugging features and the opportunity for a user to interact during this process. However, the system is not really comparable with ours since it is very specialized and dedicated to the goal of producing a fault-free alignment. Concretely, there are at least two differences to our approach. First, LogMap 2 uses incomplete reasoning mechanisms in order to speed up the matching process. Hence, the output is not guaranteed to be fault-free. Second, the option for user interaction aims in fact at the revision of a set of correspondences, i.e. the sequential assessing of single correspondences as 'faulty' or 'correct'. Our approach, on the contrary, asks the user queries (i.e. *entailments* of non-faulty parts of the KB).

An interactive technique similar to our approaches was presented in [NRG12], where a user is successively asked single KB formulas (ontology axioms) in order to obtain a partition of a given ontology into a set of desired or correct and a set of undesired or incorrect formulas. Whereas our strategies aim at finding a parsimonious solution involving minimal change to the given faulty KB in order to repair it, the method proposed in [NRG12] pursues a (potentially) more invasive approach to KB quality assurance, namely a (reasoner-supported) exhaustive manual inspection of (parts of) a KB. Given an inconsistent/incoherent KB, this technique starts from an empty set of desired formulas aiming at adding to this set only correct formulas of the KB which preserve consistency and coherency. Our approach, on the other hand, works its way forward the other way round in that it starts from the complete KB aiming at finding a minimal set of formulas to be deleted or modified which are responsible for the violation of the pre-specified requirements. Another difference of our approach compared to the one suggested in [NRG12] is the type of queries asked to the user and the way these are selected. Our method allows for the generation of queries which are not explicit formulas in the KB, but implicit consequences of non-faulty parts of the KB. Besides, the set of selectable queries in our approach differs from one iteration to the next due to the changing set of leading diagnoses whereas queries (i.e. KB formulas) in [NRG12] are known in advance and the challenge is to figure out the best ordering of formulas to be assessed by the user. Whereas we apply mostly information theoretic measures (e.g. the minimal expected entropy in the set of leading diagnoses after a query has been answered), the authors in [NRG12] employ "impact

measures" which, roughly speaking, indicate the number of automatically classifiable formulas in case of positive and, respectively, negative classification of a query (i.e. a particular formula).

Chapter 33

Summary

In this work we motivated why appropriate tool assistance is a must when it comes to repairing faulty KBs. For, KBs that do not satisfy some minimal quality criteria such as logical consistency can make artificial intelligence applications relying on the domain knowledge modeled by this KB completely useless. In such a case, no meaningful reasoning or answering of queries about the domain is possible.

Non-interactive debugging systems published in research literature often cannot localize all possible faults (*incompleteness*), suggest the deletion or modification of unnecessarily large parts of the KB (*non-minimality*), return incorrect solutions which lead to a repaired KB not satisfying the imposed quality requirements (*unsoundness*) or suffer from *poor scalability* due to the inherent complexity of the KB debugging problem [Stu08]. Even if a system is complete and sound and considers only minimal solutions, there are generally exponentially many solution candidates to select one from. However, any two repaired KBs obtained from these candidates differ in their semantics in terms of entailments and non-entailments. Selection of just any of these repaired KBs might result in unexpected entailments, the loss of desired entailments or unwanted changes to the KB which in turn might cause unexpected new faults during the further development or application of the repaired KB. Also, manual inspection of a large set of solution candidates can be time-consuming (if not practically infeasible), tedious and error-prone since human beings are normally not capable of fully realizing the semantic consequences of deleting a set of formulas from a KB.

To account for this issue, we evolved a comprehensive theory on which provably complete, sound and optimal (in terms of given probability information) interactive KB debugging systems can be built which suggest only minimal changes to repair a present KB. Interaction with a user is realized by asking the user queries. That is, a conjunction of logical formulas must be classified either as an intended or a non-intended entailment of the correct KB. To construct a query, only a minimal set of two solution candidates must be available. After the answer to a query is known, the search space for solutions is pruned. Iteration of this process until there is only a single solution candidate left yields a (repaired) solution KB which features exactly the semantics desired and expected by the user.

We presented algorithms for the computation of minimal conflict sets, i.e. irreducible faulty subsets of the KB, and for the computation of minimal diagnoses, i.e. irreducible sets of KB formulas that must be properly modified or deleted in order to repair the KB. We combined these algorithms with methods that derive probabilities of diagnoses from meta information about faults (e.g. the outcome of a statistical analysis) to constitute a non-interactive debugging system for monotonic KBs which computes minimal diagnoses in best-first order. Building on the idea of this non-interactive method, we devised a complete and sound best-first algorithm for the interactive debugging of monotonic KBs that allows a user to take part in the debugging process in order to figure out the best solution.

In order to integrate the new information collected by successive consultations of the user, the diag-

noses computation in an interactive system must be regularly stopped. That is, there must be alternating phases, on the one hand for the further exploration of the solution space in order to gain new evidence for query generation and on the other hand for user interaction. To this end, we proposed two new strategies for the iterative computation of minimal diagnoses that exactly serve this purpose. The first strategy, STATICHS, takes advantage of an artificial fixation of the solution set which guarantees the monotonic reduction of the solution space independently of the asked queries, the given answers or other parameters of the algorithm. In this vein, the complexity of this algorithm is initially known and the maximum overhead compared to the non-interactive algorithm is polynomially bound.[47] On the downside, STATICHS cannot optimally exploit the information given by the answered queries and thus cannot employ powerful methods that enable a more efficient pruning of the solution search space.

Such powerful methods can be incorporated by the second suggested strategy, DYNAMICHS, the performance of which can be orders of magnitude better than the (initially fixed) performance of STATICHS in the best case. That is, the ability to fully incorporate the information gained from user interaction might lead to a modified problem instance for which only a single (best) solution exists with only a small fraction of the time, space and user effort needed by STATICHS. Moreover, the (exact) solution located by means of an interactive debugging session applying DYNAMICHS is generally a better (verified) solution than the (exact) solution found by use of STATICHS. However, the complexity of DYNAMICHS depends to a great degree on which queries are generated and which input parameters are chosen and the worst case complexity is not initially bound as in case of STATICHS. In the design of DYNAMICHS we put a particular emphasis on memory saving behavior which is manifested, for instance, by the manner how duplicate search tree paths are handled.

For selecting the best subsequent query in interactive debugging we first proposed and exhaustively analyzed two strategies: The "split-in-half" strategy prefers queries which allow eliminating a half of the leading diagnoses. The entropy-based strategy employs information theoretic concepts to exploit knowledge about the likelihood of formulas to be faulty. Based on the probability of a formula containing an error we can predict the (expected) information gain produced by a query result, enabling us to select the best subsequent query according to a one-step-lookahead entropy-based scoring function.

In comprehensive experiments using real-world KBs we compared the entropy-based method with the "split-in-half" strategy and witnessed a significant reduction in the number of queries required to identify the correct diagnosis when the entropy-based method is applied. Depending on the quality of the given prior fault probabilities, the required number of queries could be reduced by up to 60%. In order to evaluate the robustness of the entropy-based method we experimented with different prior fault probability distributions as well as different qualities of the prior probabilities. Furthermore, we investigated cases where knowledge about fault probabilities is missing or inaccurate. In case such knowledge is unavailable, the entropy-based methods ranks the diagnoses based on the number of syntax elements contained in a formula and the number of formulas in a diagnosis. Given that this is a reasonable guess (i.e. the sought diagnosis is not at the lower end of the diagnoses ranked by their prior probabilities), the entropy-based method outperformed "split-in-half". Moreover, even if the initial guess is not reasonable, the entropy-based method improves the accuracy of the probabilities as more questions are asked. Furthermore, the applicability of the approach to real-world KBs containing thousands of formulas was demonstrated by an extensive set of evaluations.

We showed that unconditional reliance upon the entropy-based method might still be problematic in the presence of fault information that is considerably uncertain. For, the entropy-based strategy fully exploits and gains from the given fault information. In this vein, it proved to speed up the debugging procedure in the normal case. However, we found out in experiments that it might also have a negative impact on the performance in the bad case where the actual solution diagnosis is rated as highly improb-

---

[47]This holds under the reasonable assumption that, in practice, a debugging session will involve only a polynomial number of queries to an interacting user. Recall that a user can abort the debugging session at any time and select the currently most probable diagnosis as their solution to the debugging problem.

able. As an alternative, one might prefer to rely on a tool (e.g. "split-in-half") which does not consider any fault information at all. In this case, however, possibly well-chosen information cannot be exploited, resulting again in inefficient debugging actions.

Minimal effort for the interacting user can be achieved if both the query selection method is chosen carefully and the provided fault information satisfies some minimum quality requirements. In particular, for deficient fault information and unfavorable strategy for query selection, we reported on cases where the overhead in terms of user effort exceeds 2000% (!) in comparison to employing a more favorable query selection strategy. Unfortunately, assessment of the fault information is only possible a-poteriori (after the debugging session is finished and the correct solution is known). To tackle this issue, we proposed a reinforcement learning strategy (RIO) which combines the benefits of the entropy-based and the "'split-in-half' approaches, i.e. high potential (to perform well) and low risk (to perform badly). RIO continuously adapts its behavior depending on the performance achieved and in this vein minimizes the risk of integrating low-quality fault information into the debugging process.

The RIO approach makes interactive debugging practical even in scenarios where reliable fault estimates are difficult to obtain. Tested under various conditions, the RIO algorithm revealed good scalability and reaction time as well as superior average performance to both the entropy-based as well as the "split-in-half" strategy in all tested cases w.r.t. required amount of user interaction. Highest achieved savings of RIO as against the best other strategy amounted to more than 80%. Further on, the performed evaluations provided evidence that for 100% of the cases in the hardest (from the debugging point of view) class of faulty test KBs, RIO performed at least as good as the *best* other strategy and in more than 70% of these cases it even manifested superior behavior to the *best* other strategy. Choosing RIO over other approaches can involve an improvement by the factor of up to 23, meaning that more than 95% of user time and effort might be saved per debugging session.

Moreover, we came up with mechanisms for efficiently dealing with KB debugging problems involving high cardinality faults. In the standard interactive debugging approach described in the first parts of this work, the computation of queries is based on the generation of the set of *most probable (or minimum cardinality)* leading diagnoses. By this postulation, certain quality guarantees about the output solution can be given. However, we learned that dropping this requirement can bring about substantial savings in terms of time and especially space complexity of interactive debugging, in particular in debugging scenarios where faulty KBs are (partly) generated as a result of the application of automatic systems, e.g. KB (ontology) learning or matching systems.

To cope with such situations, we proposed to base query computation on *any* set of leading diagnoses using a "direct" method for diagnosis generation. Contrary to the standard method that exploits minimal conflict sets, this approach takes advantage of the duality between minimal diagnoses and minimal conflict sets and employs "inverse" algorithms to those used in the standard approach in order to determine minimal diagnoses directly from the DPI without the indirection via conflict sets.

We studied the application of this direct method to high cardinality faults in KBs and noticed that the number of required queries per debugging session is hardly affected for cases when the standard approach is also applicable. However, the direct method proved applicable and able to locate the correct solution diagnosis also in situations when the standard approach (albeit one that not yet incorporates the powerful search tree pruning techniques introduced in this work) is not due to time or memory issues.

We want to point out that this work is unique in that it provides an in-depth theoretical workup of the topic of interactive KB debugging which (to the best of our knowledge) cannot be found in such a detailed fashion in other works. Furthermore, this is the first work that gives precise definitions of the problems addressed in interactive KB debugging. Additionally, it is unique in that it features (new) algorithms that *provably* solve these interactive KB debugging problems. To account for a tradeoff between solution quality and execution time, these algorithms are equipped with a feature to compute approximate solutions where the goodness of the approximation can be steered by the user. Another unique characteristic of this work is that it deals with an *entire system* of algorithms that are required for the interactive debug-

ging of monotonic KBs, considers and details all algorithms separately, analyzes their complexity, proves their correctness and demonstrates how all these algorithms are orchestrated to make up a full-fledged and provably correct interactive KB debugging system.

Chapter 34

Future Work Topics

This work has given rise to several questions we will elaborate on in our future work:

Query Generation and Selection. Our discussions of the presented query generation methods have revealed some drawbacks (cf. Chapter 8). Albeit being a fixed-parameter tractable problem as argued, the exponential time complexity regarding the number of leading diagnoses $|\mathbf{D}|$ in case an optimal query is desired is clearly an aspect that should be improved. This high complexity arises from the paradigm of computing an optimal query w.r.t. some measure $qsm()$ by calculating a (generally exponentially large) pool $\mathbf{QP}$ of queries in a first stage, whereupon the best query in $\mathbf{QP}$ according to $qsm()$ is filtered out in a second stage.

A key to solving this issue is the use of a different paradigm that does not rely on the computation of the pool $\mathbf{QP}$. Instead, qualitative measures can be derived from quantitative measures that have been used in interactive debugging scenarios [SFFR12, RSFF13, SF10]. These qualitative measures provide a way to estimate the $qsm()$ value of *partial q-partitions*, i.e. ones where not all leading diagnoses have been assigned to the respective set in the q-partition yet. In this way a *direct* search for a query with (nearly) optimal properties is possible. A similar strategy called CKK has been employed in [SFFR12] for the information gain measure $qsm() := \mathsf{ENT}()$ (see Section 9.3). From such a technique we can expect to save a high number of reasoner calls. Because usually only a small subset of q-partitions included in a query pool (of exponential cardinality) is required to find a query with desirable properties if the search is implemented by means of a *heuristic* that involves the exploration of seemingly favorable (potential) queries and (partial) q-partitions, respectively, first.

Another shortcoming of the paradigm of query pool generation and subsequent selection of the best query is the extensive use of reasoning services which may be computationally expensive (depending on the given DPI). Instead of computing a set of common entailments Q of a set of KBs $\mathcal{K}_i^*$ first and consulting a reasoner to fill up the (q-)partition for Q in order to test whether Q is a query at all (see Chapter 8), the idea enabling a significant reduction of reasoner dependence is to compute some kind of *canonical query* without a reasoner and use simple set comparisons to decide whether the associated partition is a q-partition. Guided by qualitative properties mentioned before, a search for such q-partition with desirable properties can be accomplished *without reasoning at all*. Also, a set-minimal version of the optimal canonical query can be computed without reasoning aid. Only for the optional enrichment of the identified optimal canonical query by additional entailments and for the subsequent minimization of the enriched query, the reasoner may be employed. We will present strategies accounting for these ideas in the near future.

Another aspect that can be improved is that *only one* minimized version of each query is computed by Algorithm 4. That is, per q-partition $\mathfrak{P}$, there might be some set-minimal queries which do not occur

in the output set **QP**. From the point of view of how well a query might be understood by an interacting user, of course not all minimized queries can be assumed equally good in general. For instance, consider the minimized queries Q_4 and Q_{10} in Table 8.3 on page 113. Both are equally good regarding their q-partitions (just the sets $\mathbf{D}^+$ and $\mathbf{D}^-$ are commuted), but most people will probably agree that Q_4 is much easier to comprehend from the logical point of view and thus much easier to answer.

Hence, in order to avoid a situation where a potentially best-understood query w.r.t. $\mathfrak{P}$ is not included in **QP**, the query minimization process (see Section 8.3) might be adapted to take into account some information about faults the interacting user is prone to. This could be exploited to estimate how well this user might be able to understand and answer a query. For instance, given that the user frequently has problems to apply $\exists$ in a correct manner to express what they intend to express, but has never made any mistakes in formulating implications $\rightarrow$, then the query $Q_1 = \{\forall X\, p(X) \rightarrow q(X), r(a)\}$ might be better comprehended than $Q_2 = \{\forall X \exists Y s(X, Y)\}$. One way to achieve the finding of a well-understood query for some q-partition $\mathfrak{P}$ is to run the query minimization MINQ more than once, each time with a modified input (using a hitting set tree to accomplish this in a systematic manner – cf. Chapter 4, where an analogue idea is used to compute different minimal conflict sets w.r.t. a DPI). In this way, different set-minimal queries for $\mathfrak{P}$ can be identified and the process can be stopped when a suitable query is found.

In order to come up with such a strategy, however, one must first gain insight into how well a user might understand certain logical formalisms and what properties make a query easy to comprehend from the logical perspective. It is planned to gather corresponding data about different users in the scope of a user study and to utilize the results to achieve a model of "query hardness" (by sticking to a similar overall methodology as used in [HBP11]) in order to come up with strategies for the determination of minimal queries that are easily understood. Note that such a model could also act as a guide how to specify the initial fault probabilities of syntactical elements that are used to obtain diagnoses probabilities (see Section 4.6).

Incorporating A-Posteriori Probabilities into Diagnosis Search. As we discussed in Remark 9.3 on page 125, the a-priori ($p_{\mathbf{D},prio}()$) and the a-posteriori ($p_{\mathbf{D}}()$) diagnoses probabilities might not only differ in terms of the probability values assigned to different diagnoses, but also in terms of the probability order of diagnoses. Incorporation of updated probabilities *directly* into the hitting set tree algorithms to be used for the determination of leading diagnoses in the order prescribed by an updated probability measure is only possible if there is an additional update operator (besides Bayes' Theorem for adapting *diagnoses* probabilities) that can be applied to *formula* probabilities. For, the latter are exploited in the hitting set tree to assign probability weights to paths that are *not yet diagnoses* (cf. $p_{nodes}()$ specified by Definition 4.9 and the discussion of Formula 4.6) in order to guide the search for minimal diagnoses in best-first order. Updated *diagnosis* probabilities are not helpful at all for this purpose. Devising a reasonable mechanism of updating formula probabilities seems to be hard mostly due to the lack of suitable data that might be collected during the debugging session to accomplish that. What would be imaginable during the debugging session is to try to learn something about the fault probability of *syntactical elements* by examining the positive (all formulas are *definitely* correct) and singleton negative (the single formula is *definitely* incorrect) test cases. However, a drawback of such a strategy comes into effect when only syntactically very simple queries are used which is, for instance, the case in Example 8.1 (see the definition of the GETENTAILMENTS function there). From such queries not many useful insights concerning faulty syntactical elements might be gained. On the other hand, such queries are absolutely desirable from the point of view of how well a user might comprehend the formulas asked by the system. Hence, these two aspects seem to contradict each other. Still, it is a topic for future research to attempt to elaborate a solution for that issue.

Facilitation of More Informative User Answers. The debugging system described in this work is designed to get along with just a "minimal" feedback of a user regarding an asked query. That is, we

assume the user' answer to a query Q to be merely *true*, i.e. each formula in Q (or the conjunction of formulas in Q) must be entailed by the correct KB, or *false*, i.e. at least one formula in Q (or the conjunction of formulas in Q) must not be entailed by the correct KB. However, imagine a user being presented Q and think of how they might proceed in order to come up with an answer to Q. The first observation is that, in order to respond by *true*, a user must definitely scrutinize each single formula in Q because otherwise they could never decide for sure whether the conjunction of all formulas in Q is correct. Another observation is that a user might cease to go through the rest of the formulas in case they have already identified one that must not be an entailment of the desired KB. For, in this situation, the overall query Q is already *false*. This however indicates that at least one formula must be known to be correct or false whatever answer is given to Q. Therefore, we can usually expect a user to be able to give exactly this information, namely one formula in Q that must be incorrect, additionally to answering by *false*. This extra piece of information can be exploited to achieve better space and time efficiency in the context of diagnosis computation since knowing *which* formula must definitely not be entailed gives more information that just a set of formulas of which we know that at least one among those is not entailed. Apart from that, there might be other pieces of additional information a user might be *easily* able to give additionally to the "minimal" feedback we assume in this work. Proposing more efficient algorithms that exploit such tapes of additional information is on our future work agenda.

Usage of "Positive-Impact" Queries in Combination with DYNAMICHS.   As we discussed in Section 12.1 in the context of Algorithm 5 in dynamic mode, an added test case might give rise to some pruning steps as well as it might induce the construction of new subtrees (where "new" means that these would be no subtress of a hitting set tree w.r.t. the DPI not including this test case). The latter situation occurs when "completely new" minimal conflict sets (those that are in no subset-relationship with existing ones) are introduced by the addition of a test case. If this is the only impact of a test case, then this test case has only a negative influence on the time and space complexity of Algorithm 5 using DYNAMICHS. In other words, none of the invalidated minimal diagnoses (and no other nodes in the tree) are redundant, but all of them must additionally hit the set of "completely new" minimal conflict sets (in order to become diagnoses w.r.t. new DPI). Hence, in this case, the transition from one DPI to another including this test case results only in monotonic growth of the tree. If possible, such "negative-impact test cases" must be avoided. On the other hand, one must strive for the usage of "positive-impact test cases", i.e. those that only trigger tree pruning, but no tree expansion. Defining and studying properties that constitute such "positive-impact test cases" and "negative-impact test cases", respectively, and developing specialized algorithms for extracting exactly those types of queries that enable as substantial and effective pruning as possible in the context of DYNAMICHS is part of our already ongoing research. Note that a rough intuition of which properties make out a "positive-impact test case" is illustrated on the basis of an example in Section 12.1.

Finding the Right Expert to Answer a Query in a Collaborative KB Development Setting.   As we mentioned in Chapter 1, there are collaborative KB development projects such as the OBO Project[48] and the NCI Thesaurus[49], where many different people contribute to the specification of their knowledge in large KBs. In such a setting, it may be hard to decide who is the person that has the highest chance of being able to answer a concrete query correctly. The idea in such a scenario could be to use a combination of different measures such as educational level (e.g. professor versus PhD student) or hierarchy of contributors (e.g. senior user versus regular user), statistical information about past faults of a contributor (e.g., how many of the formulas originally authored by a person have been corrected by other persons of higher educational level) or provenance information regarding terms occurring in the query (who has

---

[48]http://obo.sourceforge.net
[49]http://nciterms.nci.nih.gov/ncitbrowser

authored most of the formulas in which these terms occur?) in order to learn an "expert model" and use it to devise some kind of recommender system [JZFF10] that suggests which person to ask a particular query.

Once established, such an expert model together with provenance information of KB formulas and other types of information discussed in Section 4.6.1 could also be exploited when it comes to the definition of the fault information provided as input to our debugging system. An example of a system which enables the remote collaborative development of KBs (ontologies) and also provides logs of interesting usage data such as formula change logs and provenance information is Web Protégé [TNNM13].

Studying the Performance of the Newly Proposed Iterative Diagnosis Computation Mechanisms.
We will conduct extensive experiments using faulty real-world KBs in order to assess the impact of the usage of the powerful search tree pruning techniques of the DYNAMICHS method or the guaranteed "convergence" towards the correct solution diagnosis of the STATICHS in comparison to interactive debugging algorithms used in our previous works [SFFR12, RSFF13, SF10, SFRF14c].

Methods for Query Selection without Computation of Diagnoses.   We are also working on "conflict-based debugging" methods that do not rely on the computation of leading diagnoses for query generation. Instead, queries might be generated directly from (minimal) conflict sets. Such methods might be used together with a boolean hitting set search tree (which was originally proposed by [JL02] and optimized by [PQ12]) where the tree is regularly pruned using test cases such that tree branching is mostly or completely suppressed. In this manner, the tree remains small in size and all in all computes only a single diagnosis, i.e. the one consistent with all answered queries. Such an approach could be very space saving. Nevertheless, it is unclear whether the number of required queries and/or the computation time might increase. Implementing such an approach and answering these open questions is a topic on our future work agenda.

Employing Advanced Reasoning Techniques to Increase Debugging Efficiency.   To cope with application contexts where reasoning is the main obstacle for efficient debugging, a plan for future work is to integrate advances reasoning techniques into our system.

For example, a modular combination of reasoners [RGH12] might be adopted. In such a system there are two sound reasoners are combined where one (R1, e.g. HermiT [SMH08]) is complete for the full logic $\mathcal{L}$ (e.g. OWL 2 [GHM$^+$08]) and the other one (R2, e.g. ELK [KKS14]) is complete for only a fragment $\mathcal{L}' \subset \mathcal{L}$ (e.g. the OWL 2 EL profile [GHM$^+$08]), but $\mathcal{L}'$ can be handles much more efficiently by R2. The system in [RGH12] could be used to assign the bulk of the workload on R2 while relying on R1 only if necessary.

Another interesting approach might be to employ techniques introduced in [GPS12] for detecting so-called "hot spots" in KBs which, when deleted from the KB, lead to much more efficient reasoning. Since reasoning in our approaches is mostly applied to fractions of the faulty KB, we could possibly benefit from such an approach. For instance, queries are entailments of a set of different non-faulty fractions $\mathcal{K}_i^* = (\mathcal{K} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup U_P$ of the original KB. Now, given that a hot spot H is included, say in $\mathcal{B} \cup U_P$, then we might well delete H from this subset of $\mathcal{K}_i^*$ and might still obtain meaningful queries. The reason is that H does not include any formulas in $U_{\mathbf{D}}$ (where $\mathbf{D}$ is the set of leading diagnoses) which are essential for query computation from the diagnosis discrimination point of view. Formulas in $\mathcal{B} \cup U_P$, on the other hand, are included in all non-faulty fractions $\mathcal{K}_i^*$ and thus do not directly serve the discrimination between diagnoses. Since $U_{\mathbf{D}}$ might be much smaller in size than $\mathcal{B} \cup U_P$ in many scenarios (due to a usually small number of leading diagnoses in $\mathbf{D}$), there might be a high chance for hot spots to be located in $\mathcal{B} \cup U_P$ rather than in $U_{\mathbf{D}}$.

Bibliography

[ARW12]   Rui Abreu, André Riboira, and Franz Wotawa. Constraint-based Debugging of Spread-sheets. In *CIbSE*, pages 1–14, 2012.

[Baa03]   Franz Baader. Appendix: Description Logic Terminology. In Franz Baader, Diego Cal-vanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *Description Logic Handbook*, pages 485–495. Cambridge University Press, 2003.

[BATJ91]   Tom Bylander, Dean Allemang, Michael Tanner, and John Josephson. The computational complexity of abduction. *Artificial Intelligence*, 49:25–60, 1991.

[BBL05]   Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope. In *IJCAI*, pages 364–369, 2005.

[BCM+07]   Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2007.

[BKP12]   Franz Baader, Martin Knechtel, and Rafael Penaloza. Context-dependent views to axioms and consequences of Semantic Web ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, 12-13:22–40, April 2012.

[BLHL+01]   Tim Berners-Lee, James Hendler, Ora Lassila, et al. The Semantic Web. 2001. http://bit.ly/18ZvAXo.

[Bor96]   Alex Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.

[BP08]   Franz Baader and R. Penaloza. Axiom Pinpointing in General Tableaux. *Journal of Logic and Computation*, 20(1):5–34, November 2008.

[CFD93]   Luca Console, Gerhard Friedrich, and Daniele Theseider Dupre. Model-Based Diagnosis Meets Error Diagnosis in Logic Programs. In *IJCAI*, pages 1494–1501, 1993.

[CGT89]   Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, I(1), 1989.

[Chu36]   Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, pages 345–363, 1936.

[CL73]   Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press Inc., 1973.

[Coo71]   Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[CP71]   John Ceraso and Angela Provitera. Sources of error in syllogistic reasoning. *Cognitive Psychology*, 2(4):400–410, 1971.

[CRV+09]   Oscar Corcho, Catherine Roussey, Vilches Blázquez, Luis Manuel, and Ivan Pérez. Pattern-based OWL Ontology Debugging Guidelines. In Eva Blomqvist, Kurt Sandkuhl, Francois Scharffe, and Vojtech Svatek, editors, *Workshop on Ontology Patterns (WOP 2009), collocated with the 8th International Semantic Web Conference (ISWC 2009).*, CEUR Workshop proceedings, pages 68–82, 2009.

[DF95]   Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[dKW87]   Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, April 1987.

[DQPS11]   Jianfeng Du, Guilin Qi, Jeff Z. Pan, and Yi-Dong Shen. A Decomposition-Based Approach to OWL DL Ontology Diagnosis. In *Proceedings of 23rd IEEE International Conference on Tools with Artificial Intelligence*, pages 659–664. IEEE Press, November 2011.

[Dur10]   Rick Durrett. *Probability: Theory and Examples, Fourth Edition*. Cambridge University Press, 2010.

[EFvH+11]   Jérôme Euzenat, Alfio Ferrara, Willem Robert van Hage, Laura Hollink, Christian Meilicke, Andriy Nikolov, Dominique Ritze, François Scharffe, Pavel Shvaiko, Heiner Stuckenschmidt, Ondrej Sváb-Zamazal, and Cássia Trojahn dos Santos. Final results of the Ontology Alignment Evaluation Initiative 2011. In *Proceedings of the 6th International Workshop on Ontology Matching*, pages 1–29. CEUR-WS.org, 2011.

[FFJS04]   Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Stumptner. Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence*, 152(2):213 – 234, 2004.

[FS05]   Gerhard Friedrich and Kostyantyn Shchekotykhin. A General Diagnosis Method for Ontologies. In Yolanda Gil, Enrico Motta, Richard Benjamins, and Mark Musen, editors, *Proceedings of the 4th International Semantic Web Conference (ISWC 2005)*, pages 232–246. Springer, 2005.

[FSW99]   Gerhard Friedrich, Markus Stumptner, and Franz Wotawa. Model-based diagnosis of hardware designs. *Artif. Intell.*, 111(1-2):3–39, 1999.

[FSZ11]   Alexander Felfernig, Monika Schubert, and Christoph Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 26(1):53–62, June 2011.

[GHM+08]   Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322, November 2008.

[GPS12]   Rafael Goncalves, Bijan Parsia, and Ulrike Sattler. Performance Heterogeneity and Approximate Reasoning in Description Logic Ontologies. In *Proceedings of 11th International Semantic Web Conference (ISWC 2012)*, pages 82–98, 2012.

[GSW89]   Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989.

[HBP11]   Matthew Horridge, Samantha Bail, and Bijan Parsia. The cognitive complexity of OWL justifications. In *Proceedings of the 10th International Semantic Web Conference (ISWC 2011)*. Springer, 2011.

[HM01]   Volker Haarslev and Ralf Müller. RACER System Description. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *1st International Joint Conference on Automated Reasoning*, volume 2083 of *Lecture Notes in Computer Science*, pages 701–705, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[Hor11]   Matthew Horridge. *Justification based Explanation in Ontologies*. PhD thesis, University of Manchester, 2011.

[HPS08]   Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and Precise Justifications in OWL. In Amit Shet, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan, editors, *Proceedings of the 7th International Semantic Web Conference (ISWC 2008)*, volume 5318 of *Lecture Notes in Computer Science*, pages 323–338. Springer, 2008.

[HPS09]   Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Lemmas for Justifications in OWL. In *Proceedings of the 22nd Workshop of Description Logics DL2009*. CEUR Workshop Proceedings, 2009.

[HPS10]   Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Justification Oriented Proofs in OWL. In *Proceedings of the 9th International Semantic Web Conference (ISWC 2010)*. Springer, 2010.

[HPS12a]   Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Extracting justifications from BioPortal ontologies. In *Proceedings of the 11th International Semantic Web Conference (ISWC 2012)*, pages 287–299, 2012.

[HPS12b]   Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Justification Masking in Ontologies. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2012.

[HSNM11]   Jakob Huber, Timo Sztyler, Jan Noessner, and Christian Meilicke. CODI: Combinatorial Optimization for Data Integration - Results for OAEI 2011. In *Proceedings of the 6th International Workshop on Ontology Matching*, 2011.

[JL99]   Philip N. Johnson-Laird. Deductive reasoning. *Annual review of psychology*, 50:109–135, 1999.

[JL02]   Yun-fei Jiang and Li Lin. Computing the minimal hitting sets with binary HS-tree. *Journal of software*, 13(12):2267–2274, 2002.

[JMSK09]   Yves R. Jean-Mary, E. Patrick Shironoshita, and Mansur R. Kabuka. Ontology Matching with Semantic Verification. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):235–251, September 2009.

[JRG11]   Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. Logmap: Logic-based and scalable ontology matching. In *Proceedings of the 10th International Semantic Web Conference (ISWC 2011)*, pages 273–288. Springer, 2011.

[JRGZH12] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, Yujiao Zhou, and Ian Horrocks. Large-scale interactive ontology matching: Algorithms and implementation. In *Proceedings of 20th European Conference on Artificial Intelligence (ECAI2012)*, pages 444–449, 2012.

[Jun04]   Ulrich Junker. QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence*, volume 3, pages 167–172. AAAI Press / The MIT Press, 2004.

[JZFF10]   Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[Kal06]   Aditya Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, University of Maryland, College Park, 2006.

[Kar72]   Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[Kaz08]   Yevgeny Kazakov. SRIQ and SROIQ are harder than SHOIQ. In *Proceedings of the 21st Workshop of Description Logics DL2008*, 2008.

[KK06]   Martin Kreuzer and Stefan Kühling. *Logik für Informatiker*. Pearson Studium, München, Germany, 2006.

[KKLO86]   Narendra Karmarkar, Richard M. Karp, George S. Lueker, and Andrew M. Odlyzko. Probabilistic analysis of optimum partitioning. *Journal of Applied Probability*, 23(3):626–645, 1986.

[KKS14]   Yevgeny Kazakov, Markus Krötzsch, and František Simančík. The incredible ELK. *Journal of automated reasoning*, 53(1):1–61, 2014.

[Kor98]   Richard E. Korf. A complete anytime algorithm for number partitioning. *Artificial Intelligence*, 106(2):181–203, December 1998.

[KPHS07]   Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all Justifications of OWL DL Entailments. In Karl Aberer, Key-Sun Choi, Natasha F. Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007*, volume 4825 of *LNCS*, pages 267–280, Berlin, Heidelberg, November 2007. Springer Verlag.

[KPS+06]   Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, and James Hendler. Swoop: A Web Ontology Editing Browser. *J. Web Sem.*, 4(2):144–153, 2006.

[KPSCG06] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca Grau. Repairing Unsatisfiable Concepts in OWL Ontologies. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006*, volume 4011 of *Lecture Notes in Computer Science*, pages 170–184, Berlin, Heidelberg, 2006. Springer.

[KPSH05]   Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging Unsatisfiable Classes in OWL Ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(4):268–293, 2005.

[MB88]   Stephen Muggleton and Wray L. Buntine. Machine Invention of First-order Predicates by Inverting Resolution. In J Laird, editor, *Proceedings of the 5th International Conference on Machine Learning (ICML'88)*, pages 339–352. Morgan Kaufmann, 1988.

[Mei11]   Christian Meilicke. *Alignment Incoherence in Ontology Matching*. PhD thesis, Universität Mannheim, 2011.

[Men09]   Elliott Mendelson. *Introduction to Mathematical Logic, Fifth Edition*. CRC Press, 2009.

[MPSP09]   Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. *W3C recommendation*, pages 1–133, 2009.

[MS72]   Albert R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on Switching and Automata Theory*, pages 125–129. IEEE, 1972.

[MS09]   Christian Meilicke and Heiner Stuckenschmidt. An Efficient Method for Computing Alignment Diagnoses. In *Proceedings of the 3rd International Conference on Web Reasoning and Rule Systems*, pages 182–196. Springer-Verlag, 2009.

[MSH09]   Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36(1):165–228, 2009.

[MST07]   Christian Meilicke, Heiner Stuckenschmidt, and Andrei Tamilin. Repairing Ontology Mappings. *Proceedings of the 22nd National Conference on Artificial intelligence - AAAI'07*, pages 1408–1413, 2007.

[MST08]   Christian Meilicke, Heiner Stuckenschmidt, and Andrei Tamilin. Reasoning Support for Mapping Revision. *Journal of Logic and Computation*, 19(5):807–829, August 2008.

[Mug95]   Stephen Muggleton. Inverse entailment and Progol 1 Introduction. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[NB12]   Duyhoa Ngo and Zohra Bellahsene. YAM++ - A combination of graph matching and machine learning approach to ontology alignment task. *Journal of Web Semantics - The Semantic Web Challenge 2011 Special Issue*, 2012.

[NCLM06]   Natalya F. Noy, A. Chugh, W. Liu, and Mark A. Musen. A framework for ontology evolution in collaborative environments. In *Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, 2006.

[NPQW13]   Iulia Nica, Ingo Pill, Thomas Quaritsch, and Franz Wotawa. The route to success: A performance comparison of diagnosis algorithms. In *Proceedings of the Twenty-Third international Joint Conference on Artificial Intelligence*, pages 1039–1045, 2013.

[NRG12]   Nadeschda Nikitina, Sebastian Rudolph, and Birte Glimm. Interactive Ontology Revision. *Web Semantics: Science, Services and Agents on the World Wide Web*, 12-13:118–130, 2012.

[NSD+00]   Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W. Fergerson, and
             Mark A. Musen. Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent
             Systems*, 16(2):60–71, 2000.

[PQ12]   Ingo Pill and Thomas Quaritsch. Optimizations for the Boolean Approach to Computing
             Minimal Hitting Sets. In *Proceedings of the 20th European Conference on Artificial Intelli-
             gence*, pages 648–653, 2012.

[PSHH+04]   Peter F. Patel-Schneider, Patrick Hayes, Ian Horrocks, et al. OWL Web Ontology Language
             Semantics and Abstract Syntax. *W3C recommendation*, 10, 2004.

[PSK05]   Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging OWL ontologies. In Allan Ellis
             and Tatsuya Hagino, editors, *Proceedings of the 14th international conference on World
             Wide Web*, pages 633–640. ACM Press, May 2005.

[PW03]   Bernhard Peischl and Franz Wotawa. Model-Based Diagnosis or Reasoning from First
             Principles. *IEEE Intelligent Systems*, 18:32–37, 2003.

[Qui86]   John Ross Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.

[RCVB09]   Catherine Roussey, Oscar Corcho, and Luis Manuel Vilches-Blázquez. A catalogue of OWL
             ontology antipatterns. In *International Conference On Knowledge Capture*, pages 205–206,
             Redondo Beach, California, USA, 2009. ACM.

[RDH+04]   Alan Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch,
             Robert Stevens, Hai Wang, and Chris Wroe. OWL Pizzas: Practical Experience of Teach-
             ing OWL-DL: Common Errors & Common Patterns. In Enrico Motta, Nigel R. Shadbolt,
             Arthur Stutt, and Nick Gibbins, editors, *Engineering Knowledge in the Age of the Seman-
             ticWeb 14th International Conference, EKAW 2004*, pages 63–81, Whittenbury Hall, UK,
             2004. Springer.

[Rei87]   Raymond Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*,
             32(1):57–95, 1987.

[RGH12]   Ana Armas Romero, Bernardo Cuenca Grau, and Ian Horrocks. MORe: Modular combina-
             tion of OWL reasoners for ontology classification. In *Proceedings of the 11th International
             Semantic Web Conference (ISWC 2012)*, 2012.

[RN10]   Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson
             Education, 3rd edition, 2010.

[Rod15]   Patrick Rodler. A Theory of Interactive Debugging of Knowledge Bases in Monotonic
             Logics. Master's thesis, Alpen-Adria Universität Klagenfurt, 2015.

[RP10]   Quentin Reul and Jeff Z. Pan. KOSIMap: Use of Description Logic Reasoning to Align
             Heterogeneous Ontologies. In Volker Haarslev, David Toman, and Grant Weddell, editors,
             Proceedings of the 23rd International Workshop on Description Logics DL2010, pages 489–
             500. CEUR Workshop Proceedings, 2010.

[RSFF11]   Patrick Rodler, Kostyantyn Shchekotykhin, Philipp Fleiss, and Gerhard Friedrich. Bal-
             ancing Brave and Cautious Query Strategies in Ontology Debugging. In Tudor Groza Vit
             Novacek, Zhisheng Huang, editor, *Proceedings of the Joint Workshop on Knowledge Evolu-
             tion and Ontology Dynamics 2011 (EvoDyn2011)*, Bonn, Germany, 2011. CEUR Workshop
             Proceedings.

[RSFF12]   Patrick Rodler, Kostyantyn Shchekotykhin, Philipp Fleiss, and Gerhard Friedrich. RIO: Minimizing User Interaction in Debugging of Aligned Ontologies. In *Proceedings of the 7th International Workshop on Ontology Matching (OM-2012)*, 2012.

[RSFF13]   Patrick Rodler, Kostyantyn Shchekotykhin, Philipp Fleiss, and Gerhard Friedrich. RIO: Minimizing User Interaction in Ontology Debugging. In Wolfgang Faber and Domenico Lembo, editors, *Web Reasoning and Rule Systems*, volume 7994 of *Lecture Notes in Computer Science*, pages 153–167. Springer Berlin Heidelberg, 2013.

[SE13]   Pavel Shvaiko and Jérôme Euzenat. Ontology matching: State of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):158–176, 2013.

[SEA$^+$02]   York Sure, Michael Erdmann, Juergen Angele, Steffen Staab, Rudi Studer, and Dirk Wenke. OntoEdit: Collaborative Ontology Development for the Semantic Web. In *Proceedings of the 1st International Semantic Web Conference (ISWC 2002)*, pages 221–235, 2002.

[Set12]   Burr Settles. *Active Learning*. Morgan and Claypool Publishers, 2012.

[SF10]   Kostyantyn Shchekotykhin and Gerhard Friedrich. Query strategy for sequential ontology debugging. In Peter F. Patel-Schneider, Pan Yue, Pascal Hitzler, Peter Mika, Zhang Lei, Jeff Pan, Ian Horrocks, and Birte Glimm, editors, *Proceedings of the 9th International Semantic Web Conference (ISWC 2010)*, pages 696–712, Shanghai, China, 2010.

[SFFR12]   Kostyantyn Shchekotykhin, Gerhard Friedrich, Philipp Fleiss, and Patrick Rodler. Interactive Ontology Debugging: Two Query Strategies for Efficient Fault Localization. *Web Semantics: Science, Services and Agents on the World Wide Web*, 12-13:88–103, 2012.

[SFJ08]   Kostyantyn Shchekotykhin, Gerhard Friedrich, and Dietmar Jannach. On Computing Minimal Conflicts for Ontology Debugging. In *MBS 2008 - Workshop on Model-Based Systems*, 2008.

[SFRF12]   Kostyantyn Shchekotykhin, Philipp Fleiss, Patrick Rodler, and Gerhard Friedrich. Direct computation of diagnoses for ontology alignment. In Pavel Shvaiko, Jérôme Euzenat, Anastasios Kementsietsidis, Ming Mao, Natasha Noy, and Heiner Stuckenschmidt, editors, *Proceedings of the 7th International Workshop on Ontology Matching (OM2012)*, pages 244–245, Boston, MA USA, 2012. CEUR Workshop Proceedings.

[SFRF14a]   Kostyantyn Shchekotykhin, Gerhard Friedrich, Patrick Rodler, and Philipp Fleiss. A direct approach to sequential diagnosis of high cardinality faults in knowledge bases. In *DX 2014 - 25th International Workshop on Principles of Diagnosis (DX 2014)*, 2014.

[SFRF14b]   Kostyantyn Shchekotykhin, Gerhard Friedrich, Patrick Rodler, and Philipp Fleiss. Interactive Ontology Debugging using Direct Diagnosis. In Patrick Lambrix, Guilin Qi, Matthew Horridge, and Bijan Parsia, editors, *Proceedings of the Third International Workshop on Debugging Ontologies and Ontology Mappings (WoDOOM14)*. CEUR Workshop Proceedings, 2014.

[SFRF14c]   Kostyantyn Shchekotykhin, Gerhard Friedrich, Patrick Rodler, and Philipp Fleiss. Sequential diagnosis of high cardinality faults in knowledge-bases by direct diagnosis generation. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*. IOS Press, 2014.

[Sha48]   Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[Sha83]   Ehud Shapiro. *Algorithmic Program Debugging*. MIT Press, 1983.

[SHCH07]   Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank Harmelen. Debugging Incoherent Terminologies. *Journal of Automated Reasoning*, 39(3):317–349, 2007.

[SKFP12]   Roni Stern, Meir Kalech, Alexander Feldman, and Gregory Provan. Exploring the Duality in Conflict-Directed Model-Based Diagnosis. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence Exploring*, pages 828–834, 2012.

[SL89]   Bart Selman and Hector Levesque. Abductive and default reasoning: A computational core. *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 343–348, 1989.

[SMH08]   Rob Shearer, Boris Motik, and Ian Horrocks. HermiT : A Highly-Efficient OWL Reasoner. In *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*, 2008.

[SPG$^+$07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Y Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007.

[SQJH08]   Boontawee Suntisrivaraporn, Guilin Qi, Qiu Ji, and Peter Haase. A Modularization-Based Approach to Finding All Justifications for OWL DL Entailments. In *Proceedings of the 7th International Semantic Web Conference (ISWC 2008)*, pages 1–15. Springer, 2008.

[SRF11]   Kostyantyn Shchekotykhin, Patrick Rodler, and Gerhard Friedrich. Balancing brave and cautious query strategies in ontology debugging. In *22nd International Workshop on Principles of Diagnosis (DX 2011)*, pages 122–129, 2011.

[SS89]   Manfred Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 421–431. Morgan Kaufmann Publishers Inc., 1989.

[SSZ09]   Ulrike Sattler, Thomas Schneider, and Michael Zakharyaschev. Which Kind of Module Should I Extract? In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Proceedings of the 22nd International Workshop on Description Logics*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[Stu08]   Heiner Stuckenschmidt. Debugging OWL Ontologies - A Reality Check. In Raul Garcia-Castro, Asunción Gómez-Pérez, Charles J. Petrie, Emanuele Della Valle, Ulrich Küster, Michal Zaremba, and Shafiq M. Omair, editors, *Proceedings of the 6th International Workshop on Evaluation of Ontology-based Tools and the Semantic Web Service Challenge (EON)*, pages 1–12, Tenerife, Spain, 2008.

[SU06]   Ken Satoh and Takeaki Uno. Enumerating Minimally Revised Specifications Using Dualization. In Takashi Washio, Akito Sakurai, Katsuto Nakajima, Hideaki Takeda, Satoshi Tojo, and Makoto Yokoo, editors, *New Frontiers in Artificial Intelligence*, volume 4012 of *Lecture Notes in Computer Science*, pages 182–189. Springer Berlin Heidelberg, 2006.

[SW05]   Gerald Steinbauer and Franz Wotawa. Detecting and locating faults in the control software of autonomous mobile robots. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 1742–1743, 2005.

[SW09]   Gerald Steinbauer and Franz Wotawa. Robust Plan Execution Using Model-Based Reasoning. *Advanced Robotics*, 23(10):1315–1326, 2009.

[TH06]   Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *In Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, pages 292–297. Springer, 2006.

[TNNM13]   Tania Tudorache, Csongor Nyulas, Natalya F. Noy, and Mark A. Musen. WebProtégé: A Collaborative Ontology Editor and Knowledge Acquisition Tool for the Web. *Semantic Web*, 4(1):89–99, 2013.

[Tur37]   Alan Mathison Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(1):230–265, 1937.

[WSM02]   Franz Wotawa, Markus Stumptner, and Wolfgang Mayer. Model-Based Debugging or How to Diagnose Programs Automatically. In Tim Hendtlass and Moonis Ali, editors, *Developments in Applied Artificial Intelligence*, volume 2358 of *Lecture Notes in Computer Science*, pages 746–757. Springer Berlin Heidelberg, 2002.