QuickXPlain for Everyone: Simple Explanation and Formal Proof

Patrick Rodler

University of Klagenfurt
Universitätsstrasse 65-67, 9020 Klagenfurt

Abstract

In his seminal paper of 2004, Ulrich Junker proposed the QuickXPlain algorithm, which provides a divide-and-conquer computation strategy to find within a given set an irreducible subset with a particular (monotone) property. Beside its original application in the domain of constraint satisfaction problems, the algorithm has since then found widespread adoption in areas as different as model-based diagnosis, recommender systems, verification, or the Semantic Web. This popularity is due to the frequent occurrence of the problem of finding irreducible subsets on the one hand, and to QuickXPlain's general applicability and favorable computational complexity on the other hand.

However, although (we regularly experience) people are having a hard time understanding QuickXPlain and seeing why it works correctly, a proof of correctness of the algorithm has never been published. This is what we account for in this work, by explaining QuickXPlain in a novel tried and tested way and by presenting an intelligible formal proof of it. Apart from showing the correctness of the algorithm and excluding the later detection of errors (*proof and trust effect*), the added value of the availability of a formal proof is, e.g., *(i)* that the workings of the algorithm often become completely clear only after studying, verifying and comprehending the proof (*didactic effect*), *(ii)* the shown proof methodology can be used as a guidance for proving other recursive algorithms (*transfer effect*), and *(iii)* the possibility of providing "gapless" correctness proofs of systems that rely on (results computed by) QuickXPlain, such as numerous model-based debuggers (*completeness effect*).

Keywords: QuickXPlain, Correctness Proof, Proof to Explain, Algorithm, Find Irreducible Subset with Monotone Property, MSMP Problem, Minimal Unsatisfiable Subset, Minimal Correction Subset, Model-Based Diagnosis, CSP

1. Introduction

The task of finding within a given universe an irreducible subset with a specific monotone property is referred to as the *MSMP* (*M*inimal *S*et subject to a *M*onotone *P*redicate) *problem* [1, 2]. Take the set of clauses $S := \{\neg C, A \vee$

$\neg B, C \vee \neg B, \neg A, B\}$ as an example. This set is obviously unsatisfiable. One task of interest expressible as an MSMP problem is to find a minimal unsatisfiable subset (MUS) of these clauses (which can help, e.g., to understand the cause of the clauses' inconsistency). At this, S is the *universe*, and the predicate that tells whether a given set of clauses is satisfiable is *monotone*, i.e., any superset (subset) of an unsatisfiable (satisfiable) clause set is unsatisfiable (satisfiable). In fact, there are two MUSes for S, i.e., $\{\neg C, C \vee \neg B, B\}$ and $\{A \vee \neg B, \neg A, B\}$. We call a task, such as MUS, that can be formulated as an MSMP problem a *manifestation of the MSMP problem*.

MSMP is relevant to a wide range of computer science disciplines, including model-based diagnosis [3, 4, 5, 6], constraint satisfaction problems [7, 8, 9], verification [1, 10, 11, 12], configuration problems [13, 14], knowledge representation and reasoning [15, 16, 17, 18], recommender systems [19, 20], knowledge integration [21, 22], as well as description logics and the Semantic Web [5, 23, 24, 25, 26, 27]. In all these fields, (sub)problems are addressed which are manifestations of the MSMP problem. Example problems—most of them related to the Boolean satisfiability problem—are the computation of *minimal unsatisfiable subsets* [2, 28, 29, 30] (also termed *conflicts* [31, 32] or *minimal unsatisfiable cores* [28]), *minimal correction subsets* [33, 34, 35] (also termed *diagnoses* [31, 32]), *prime implicants* [36, 37] (also termed *justifications* [25]), *prime implicates* [18, 38, 39], and *most concise optimal queries to an oracle* [21, 27, 40, 41].

Numerous algorithms to solve manifestations of the MSMP problem have been suggested in literature, e.g., [1, 2, 4, 7, 8, 10, 41, 42, 43, 44, 45]. For instance, the algorithm proposed by Felfernig et al. [44] addresses the problem of the computation of minimal correction subsets (diagnoses), and the one suggested by Rodler et al. [41] computes minimal oracle queries that preserve some optimality property. In general, an algorithm A for a specific manifestation of the MSMP problem can be used to solve arbitrary manifestations of the MSMP problem if *(i)* the procedure used by A to decide the monotone predicate is used as a black-box (i.e., given a subset of the universe as input, the procedure outputs 1 if the predicate is true for the subset and 0 otherwise; no more and no less), and *(ii)* no assumptions or additional techniques are used in A which are specific to one particular manifestation of the MSMP problem.

Not all algorithms meet these two criteria. For instance, there are algorithms that rely on additional outputs beyond the mere evaluation of the predicate (e.g., certificate-refinement-based algorithms [2]), or glass-box approaches that use non-trivial modifications of the predicate decision procedure to solve the MSMP problem (e.g., theorem provers that record the axioms taking part in the deduction of a contradiction while performing a consistency check [5]). These methods violate (i). Moreover, e.g., algorithms geared to the computation of minimal unsatisfiable subsets that leverage a technique called model rotation [46] are not applicable, e.g., to the problem of finding minimal correction subsets, since there is no concept equivalent to model rotation for minimal correction subsets [2]. Thus, such algorithms violate (ii).

Among the general MSMP algorithms that satisfy (i) and (ii), QuickX-

PLAIN [8] (QX for short), proposed by Ulrich Junker in 2004, is one of the most popular and most frequently adopted.[1] Likely reasons for the widespread use of QX are its mild theoretical complexity in terms of the number of (usually expensive[2]) predicate evaluations required [2, 8], as well as its favorable practical performance for important problems (such as conflict [47] or diagnosis [42] computation for model-based diagnosis). In literature, QX is utilized in different ways; it is *(a) (re)used as is* for suitable manifestations of MSMP [13], *(b) adapted* in order to solve other manifestations of MSMP [4], as well as *(c) modified or extended*, respectively, e.g., to achieve a better performance for a particular MSMP manifestation [2], to solve extensions of the MSMP problem [41], or to compute multiple minimal subsets of the universe in a single run [43].

Despite its popularity and common use, from the author's experience,[3] QX appears to be quite poorly understood by reading and thinking through the algorithm, and, for most people, requires significant and time-consuming attention until they are able to properly explain the algorithm. In particular, people often complain they do not see why it correctly computes a minimal subset of the universe. This is not least because no proof of QX has yet been published.

In this work, we account for this by presenting a clear and intelligible proof of QX. The public availability of a proof comes with several benefits and serves i.a. the following purposes:

Proof Effect *(a)* It shows QX's correctness and makes it verifiable for everyone in a straightforward step-by-step manner (without the need to accomplish the non-trivial task of coming up with an own proof). *(b)* It creates compliance with common scientific practice. That is, every proposal of an algorithm should be accompanied with a (full and public) formal proof of correctness. This demand is even more vital for a highly influential algorithm like QX.

Didactic Effect *(a)* It promotes (proper and full) understanding [48] of the workings of QX, which is otherwise for many people only possible in a laborious way (e.g., by noting down and exercising through examples and attempting to verify QX's soundness on concrete cases). *(b)* It provides the basis for understanding (hundreds of) other works or algorithms that use, rely on, adapt, modify or extend QX.

Completeness Effect It is necessary to establish and prove the full correctness

[1] Judged by taking the citation tally on Google Scholar as a criterion; as of January 2020, the QUICKXPLAIN paper boasts 420 citations.

[2] In many manifestations of the MSMP problem, predicate decision procedures are implemented by theorem provers, e.g., SAT-solvers [2] or description logic reasoners [4].

[3] In our research and teaching on model-based diagnosis, we frequently discuss and analyze QX—one of the core algorithms used in our works and prototypes—with students as well as other faculty (including highly proficient university professors specialized in, e.g., algorithms and data structures). The feedback of people is usually that they cannot fully grasp the workings of QX before they take significant time to go through a particular example thoroughly and noting down all single steps of the algorithm. According to people's comments, the main obstacle appears to be the recursive nature of the algorithm.

of other algorithms that rely on (the correctness of) QX, such as a myriad of algorithms in the field of model-based diagnosis.

Trust and Sustainability Effect It excludes the possibility of the (later) detection of flaws in the algorithm, and is thus the only basis for placing full confidence in the proper-functioning of QX.[4]

Transfer Effect It showcases a stereotypic proof concept for recursive algorithms and can provide guidance to researchers when approaching the (often challenging task of formulating a) proof of other recursive algorithms.

The rest of this paper is organized as follows. We discuss related work in Sec. 2, before we briefly introduce the theoretical concepts required for the understanding and proof of QX in Sec. 3. Then, in Sec. 4, we state the QX algorithm in a (slightly) more general formulation than originally published in [8], i.e., we present QX as a general method to tackle the MSMP problem.[5] In addition, we explain the functioning of QX, and present an illustrative example using a notation that proved particularly comprehensible in our experience.[6] The proof is given in Sec. 5. In Sec. 6 we explain the general proof template we used which can be a helpful tool to prove other recursive algorithms as well. Concluding remarks are made in Sec. 7.

2. Related Work

Bradley and Manna [10] state an algorithm claimed to be equivalent to QX and give a proof of this algorithm. However, first, there is no proof that the stated algorithm is indeed equivalent to QX (which is not clear from the formulation given in [10]). Second, the proof given in [10] does not appear to be of great help to better understand QX, as the reader needs to become familiar with the notation and concepts used in [10] in the first place, and needs to map the pseudocode notation of [10] to the largely different one stuck to by Junker in the original QX-paper [8]. Apart from that, the proof in [10]—despite (or perhaps exactly because of) its undeniable elegance—is not "operation-centric" in that it

[4] A prominent example which shows that even seminal papers are not charmed against errors in absence of formal proofs, and thus underscores the importance of (public) proofs, is the highly influential paper of Raymond Reiter from 1987 [31]. It proposes the hitting set algorithm for model-based diagnosis, but omits a formal proof of correctness. And, indeed, a critical error in the algorithm was later found (and corrected) by Russell Greiner et al. [49].

[5] The original algorithm was depicted specifically as a searcher for explanations or relaxations for over-constrained constraint satisfaction problems (CSPs). Although the proper interpretation of the original formulation to address arbitrary MSMP problems different from CSPs may be relatively straightforward (for people familiar with CSPs), we believe that our more general depiction (cf. [2]) can help readers non-familiar with the domain of CSPs to understand and correctly use QX without needing to properly re-interpret concepts from an unknown field.

[6] We (informally) experimented with different variants how to explain QX, and found out (through the feedback of discussion partners, e.g., students) that the shown representation was more accessible than others.

is not amenable to a mental "tracking" by means of the call-recursion-tree produced by QX. In contrast, our proof is *illustrative* as it can be viewed as directly traversing the call-recursion-tree (cf. Fig. 1 later), while showing that certain invariant statements remain valid through all transitions in the tree, and using these invariants to prove that all (recursive) calls work correctly. Moreover, we segment our proof into small, intuitive, and easily digestible chunks, thus putting a special *focus on its clarity, elucidation, and didactic value*. Finally, our proof enables the verification of the correctness of the *original formulation of* QX. Hence, we believe that our proof is more valuable to people having a hard time understanding QX than the one in [10]. Or, to put it into the words of Hanna [50, 51], we present a *proof that explains*, rather than one that solely proves.

3. Basics

QX can be employed to find, for an input set U, a minimal[7] subset $X \subseteq U$ that has a certain monotone property p. An example would be an (unsatisfiable) knowledge base (set of logical sentences) U for which we are interested in finding a minimal unsatisfiable subset (MUS) X.

Definition 1 (Monotone Property)**.** *Let U be the universe (a set of elements) and $p : 2^U \rightarrow \{0,1\}$ be a function where $p(X) = 1$ iff property p holds for $X \subseteq U$. Then, p is a monotone property iff $p(\emptyset) = 0$ and*

$$\forall X', X'' \subseteq U : \; X' \subset X'' \implies p(X') \leq p(X'')$$

So, p is monotone iff, given that p holds for some set X', it follows that p also holds for any superset X'' of X'. An equivalent definition is: If p does not hold for some set X'', p does not hold for any subset X' of X'' either.

In practical applications it is often a requirement that *(a)* some elements of the universe must not occur in the sought minimal subset, or *(b)* the minimal subset of the universe should be found in the context of some reference set. Both cases (a) and (b) can be subsumed as searching for a minimal subset of the *analyzed set* $\mathcal{A}$ given some *background* $\mathcal{B}$. In case (a), $\mathcal{B}$ is defined as a subset of the universe U (e.g., in a fault localization task, those sentences of a knowledge base U that are assumed to be correct) and $\mathcal{A}$ is constituted by all other elements of the universe $U \setminus \mathcal{B}$ (those sentences in U that are possibly faulty); in case (b), $\mathcal{B}$ is some additional set of relevance to the universe (e.g., a knowledge base of general medical knowledge), whereas $\mathcal{A}$ is the universe itself (e.g., a knowledge base describing a medical sub-discipline). For example, the problem of finding a MUS wrt. $\mathcal{A}$ given background $\mathcal{B}$ would be to search for a minimal set X of elements in $\mathcal{A}$ such that $X \cup \mathcal{B}$ is unsatisfiable.

[7] Throughout this paper, minimality always refers to minimality wrt. set-inclusion.

Algorithm 1 QX: Computation of a Minimal p-Set

Input: a p-PI $\langle \mathcal{A}, \mathcal{B} \rangle$ where $\mathcal{A}$ is the analyzed set and $\mathcal{B}$ is the background
Output: a minimal p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$, if existent; 'no p-set', otherwise

```
1: procedure QX(⟨A, B⟩)
2:     if p(A ∪ B) = 0 then
3:         return 'no p-set'
4:     else if A = ∅ then
5:         return ∅
6:     else
7:         return QX′(B, ⟨A, B⟩)

8: procedure QX′(Δ, ⟨A, B⟩)
9:     if Δ ≠ ∅ ∧ p(B) = 1 then
10:        return ∅
11:    if |A| = 1 then
12:        return A
13:    k ← SPLIT(|A|)
14:    A1 ← GET(A, 1, k)
15:    A2 ← GET(A, k + 1, |A|)
16:    X2 ← QX′(A1, ⟨A2, B ∪ A1⟩)
17:    X1 ← QX′(X2, ⟨A1, B ∪ X2⟩)
18:    return X1 ∪ X2
```

Definition 2 (p-Problem-Instance)**.** *Let $\mathcal{A}$ (analyzed set) and $\mathcal{B}$ (background) be (related) finite sets of elements where $\mathcal{A} \cap \mathcal{B} = \emptyset$, and let p be a monotone predicate. Then we call the tuple $\langle \mathcal{A}, \mathcal{B} \rangle$ a p-problem-instance (p-PI).*

Definition 3 (Minimal p-Set (given some Background))**.** *Let $\langle \mathcal{A}, \mathcal{B} \rangle$ be a p-PI. Then, we call X a* p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$ *iff $X \subseteq \mathcal{A}$ and $p(X \cup \mathcal{B}) = 1$. We call a p-set X wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$* minimal *iff there is no p-set $X' \subset X$ wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$.*

Immediate consequences of Defs. 1 and 3 are:

Proposition 1 (Existence of a p-Set)**.**

(1) A (minimal) p-set exists for $\langle \mathcal{A}, \mathcal{B} \rangle$ iff $p(\mathcal{A} \cup \mathcal{B}) = 1$.
(2) $\emptyset$ is a—and the only—(minimal) p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$ iff $p(\mathcal{B}) = 1$.[8]

[8] Cf. (2) with Prop. 5 (unproven) in [8].

4. Brief Review and Explanation of QX

The QX algorithm is depicted by Alg. 1.[9] It gets as input a p-PI $\langle \mathcal{A}, \mathcal{B} \rangle$ and assumes a sound and complete oracle that answers queries of the form $p(X)$ for arbitrary $X \subseteq \mathcal{A} \cup \mathcal{B}$. If existent, QX returns a minimal p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$; otherwise, 'no p-set' is output. In a nutshell, QX works as follows:

Trivial Cases: Before line 7 is reached, the algorithm checks if trivial cases apply, i.e., if either no p-set exists (line 2; cf. Prop. 1.(1)) or a p-set does exist and the analyzed set $\mathcal{A}$ is empty (line 4), and returns according outputs. In case the execution reaches line 7, the recursive procedure QX$'$ is called. In the very first execution of QX$'$, the presence of two other trivial cases for the original input p-PI $\langle \mathcal{A}, \mathcal{B} \rangle$ is checked in lines 9 and 11.[10] (Line 9): If $p(\mathcal{B}) = 1$, then the empty set, the only minimal p-set in this case (cf. Prop. 1.(2)), is directly returned and QX terminates.[11] Otherwise, we know the empty set is not a p-set, i.e., every (minimal) p-set is non-empty. (Line 11): If the analyzed set $\mathcal{A}$ is a singleton, then $\mathcal{A}$ is directly returned and QX terminates.

Recursion: Subsequently, the recursion is started. The principle is to partition the analyzed set $\mathcal{A} = \{a_1, \ldots, a_{|\mathcal{A}|}\}$ into two *non-empty* (e.g., equal-sized) subsets $\mathcal{A}_1 = \{a_1, \ldots, a_k\}$ and $\mathcal{A}_2 = \{a_{k+1}, \ldots, a_{|\mathcal{A}|}\}$ (SPLIT and GET functions; lines 13–15), and to analyze these subsets recursively (*divide-and-conquer*). In this vein, a binary call-recursion-tree is built (as sketched by the grayscale part of Fig. 1), including the *root* QX$'$-call made in line 7 and *two subtrees*, the left one rooted at the call of QX$'$ in line 16 which analyzes $\mathcal{A}_2$, and the right one rooted at the call of QX$'$ in line 17 which analyzes $\mathcal{A}_1$. Let the finally returned minimal p-set be denoted by X, and let us call all elements of X *relevant*, all others *irrelevant*. Then, the left subtree (finally) returns the subset of those elements (X_2) from $\mathcal{A}_2$ that belong to X, and the right subtree (finally) returns the subset of those elements (X_1) from $\mathcal{A}_1$ that belong to X.

- *Arguments of the recursive procedure* QX$'$*:* The arguments $\Delta, \langle \mathcal{A}, \mathcal{B} \rangle$ passed to the procedure QX$'$ can be intuitively understood as follows. $\langle \mathcal{A}, \mathcal{B} \rangle$ is the

[9] The analyzed set, denoted by $\mathcal{A}$ in Alg. 1, is referred to by $\mathcal{C}$ in the original algorithm in [8]. Moreover, for simplicity, we omit in the presented algorithm the *explicit* statement that elements of the analyzed set $\mathcal{A}$ are sorted according to some preference order. Actually, this preference order *(1)* is irrelevant for the (soundness and termination) proof presented, which proves that, if existent, QX will find *some* minimal p-set, and instead *(2)* determines *which* (of possibly multiple) minimal p-sets will be found (see [8, 41] for details).

[10] Clearly, the checks in lines 9 and 11 are executed in every recursive QX$'$-call. However, in all further recursive calls, these tests serve to determine whether a particular given *subset* of the analyzed set must be further processed or not. We separately discuss the first two such checks *related to the original input problem* in this paragraph in order to cover *all* trivial cases before moving on to elucidate the recusion.

[11] Remarks: *(1)* $\Delta = \mathcal{B}$ in the very first execution of QX$'$, cf. lines 7 and 8. *(2)* $p(\mathcal{B}) = 1$ implies that the first condition $\Delta \neq \emptyset$ checked in line 9 is true as well (cf. Def. 1). *(3)* Actually, the check in line 9 in the very first execution of QX$'$ complements the one in line 4. The reason is that in line 4 $\mathcal{A} = \emptyset$ and $p(\mathcal{B}) = 1$ (due to line 2) holds, whereas in line 9 $\mathcal{A} \neq \emptyset$ and $p(\mathcal{B}) = 1$ is true.

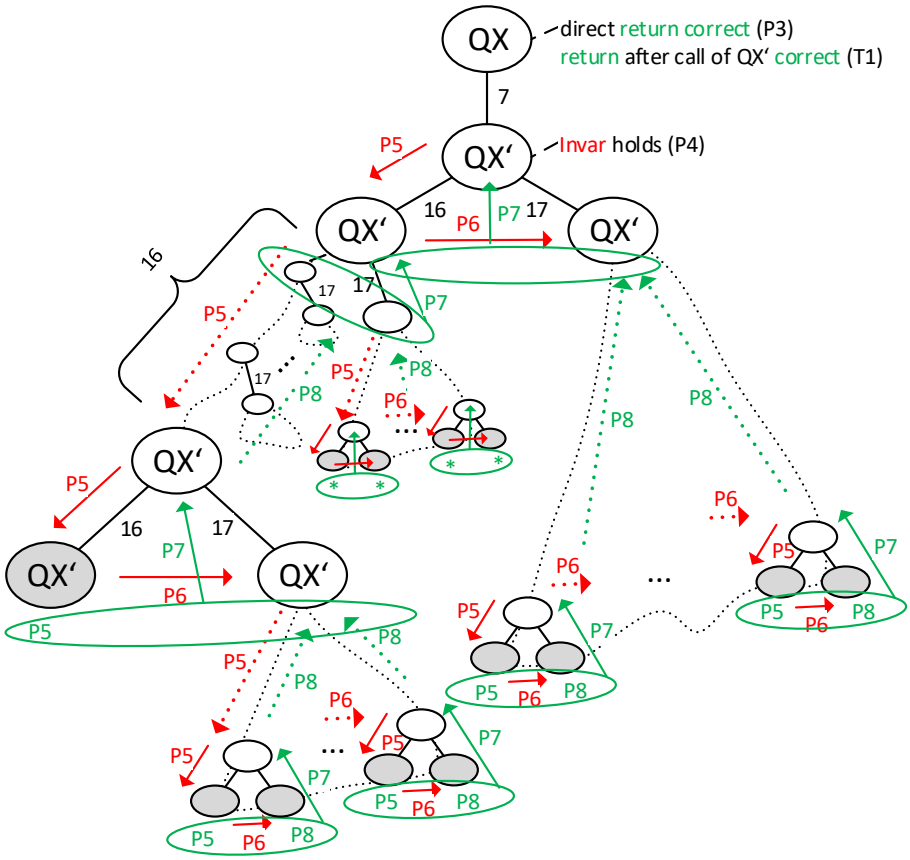

Figure 1: Call-recursion-tree produced by QX (cf. Sec. 4). The *grayscale part of the figure* provides a schematic illustration of the procedure calls executed in a single run of QX (where the recursion is entered, i.e., no trivial case applies). Each node (ellipse) represents one call of the procedure named within the ellipse. Edge labels (7,16,17) refer to the lines in Alg. 1 where the respective call is made. White ellipses (non-leaf nodes) are calls that issue further recursive calls (in lines 16 and 17), whereas gray ellipses (leaf nodes) are calls that directly return (i.e., in line 10 or 12). The *colored part of the figure* visualizes the meaning and consequences of the theorem (T1) and the various propositions (Pi, for $i \in \{3, 4, 5, 6, 7, 8\}$) that constitute the proof (cf. Sec. 5). Red arrows indicate proven propagations of the invariant property Invar (see Def. 4) between calls. Green arrows and labels indicate that respective calls return correct outputs. Start to read the colored illustrations from the top, just like QX proceeds. That is, due to P3, direct returns yield correct outputs. If QX′ is called, Invar holds by P4. If Invar holds for some call, then it is always propagated downwards to the left subtree because of P5. At the first leaf node, a correct output is returned, also due to P5. If the output of a left subtree is correct, then Invar propagates to the right subtree (P6). If the output of both the left and the right subtree is correct, then the output of the root is correct (P7). If Invar holds at the root call of some (sub)tree, then this root call returns a correct output (P8). Note how these propositions guarantee that Invar, and thus correct outputs, can be derived for all nodes of the call-recursion-tree. Intuitively, red arrows propagate Invar downwards through the tree, which then ensures correct outcomes at the leafs, from where these correct outputs enable further propagation of Invar to the right, from where the inferred correct outputs are recursively propagated upwards until the root node is reached.

p-PI analyzed by the respective QX′-call. Δ is (only[12]) relevant when QX′ was called in line 17 and essentially indicates whether some relevant element was found while analyzing $\mathcal{A}_2$ in the left subtree (QX′-call in line 16). If so ($\Delta = X_2 \neq \emptyset$), then Δ "activates" the test ($p(\mathcal{B}) = 1$?) in line 9 that checks if an exploration of the right subtree ($\mathcal{A}_1$) is superfluous (or, in other words, if a full minimal p-set is already contained in $\mathcal{A}_2$). Otherwise ($\Delta = X_2 = \emptyset$; no relevant element in $\mathcal{A}_2$), Δ "deactivates" this check since a relevant element *must* be included in $\mathcal{A}_1$ (because at least one of $\mathcal{A}_1$ and $\mathcal{A}_2$ must include a relevant element), and therefore returning $\emptyset$ in line 10 as a result for $\mathcal{A}_1$ must be precluded.

- *Left subtree (recursive* QX′*-call in line 16):* The first question is: Are all elements of $\mathcal{A}_2$ irrelevant? Or, equivalently: Does $\mathcal{B} \cup \mathcal{A}_1$ already contain a minimal p-set, i.e., $p(\mathcal{B} \cup \mathcal{A}_1) = 1$? This is evaluated in line 9; note: $\Delta = \mathcal{A}_1 \neq \emptyset$. If positive, $\emptyset$ is returned and the subtree is not further expanded. Otherwise, we know there is some relevant element in $\mathcal{A}_2$. Hence, the analysis of $\mathcal{A}_2$ is started. That is, in line 11, the singleton test is performed for $\mathcal{A}_2$. In the affirmative case, we have proven that the single element in $\mathcal{A}_2$ is relevant. The reason is that $p(\mathcal{B} \cup \mathcal{A}_1) = 0$, as verified in line 9 just before, and that adding the single element in $\mathcal{A}_2$ makes the predicate true,[13] i.e., $p(\mathcal{B} \cup \mathcal{A}_1 \cup \mathcal{A}_2) = p(\mathcal{B} \cup \mathcal{A}) = 1$, as verified in line 2 at the very beginning. If $\mathcal{A}_2$ is a non-singleton, it is again partitioned and the subsets are analyzed recursively, which results in two new subtrees in the call-recursion-tree.

- *Right subtree (recursive* QX′*-call in line 17):* Here, we can distiguish between two possible cases, i.e., either the set X_2 returned by the left subtree is *(i)* empty or *(ii)* non-empty.
  Given (i), we know that $\mathcal{A}_1$ must include a relevant element. Reason: $\mathcal{B} \cup \mathcal{A}_1$ contains a minimal p-set (as verified in the left subtree before returning the empty set) and every p-set is non-empty (as verified in line 9 in the course of checking the *Trivial Cases*, see above). Hence, $\mathcal{A}_1$ is further analyzed in lines 11 et seqq. (which might lead to a direct return if $\mathcal{A}_1$ is a singleton and thus relevant, or to further recursive subtrees otherwise).
  For (ii), the question is: Given the subset X_2 of the p-set, are all elements of $\mathcal{A}_1$ irrelevant? Or, equivalently: Does $\mathcal{B} \cup X_2$ already contain a minimal p-set, i.e., $p(\mathcal{B} \cup X_2) = 1$? This is answered in line 9; note: $\Delta = X_2 \neq \emptyset$ due to case (ii). In the affirmative case, the empty set is returned, i.e., no elements of $\mathcal{A}_1$ are relevant and the final p-set X found by QX is equal to X_2. If the answer is negative, $\mathcal{A}_1$ does include some relevant element and is thus further analyzed in lines 11 et seqq. (which might lead to a direct return if $\mathcal{A}_1$ is a singleton and thus relevant, or to further recursive subtrees otherwise).

[12] If QX′ was called in line 7, the truth of the condition tested in line 9 depends only on $p(\mathcal{B})$ (cf. Footnote 11) and thus Δ has no effect. Similarly, given that QX′ was called in line 16, Δ always corresponds to the non-empty set $\mathcal{A}_1$ and, again, $p(\mathcal{B})$ alone determines the truth value of the condition in line 9.

[13] Such an element is commonly referred to as a *necessary* or a *transition* element [45].

Finally, the union of the outcomes of left (X_2) and right (X_1) subtrees is a minimal p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$ and returned in line 18.

Example 1 We illustrate the functioning of QX by means of a simple example.

Input Problem and Parameter Setting: Assume the analyzed set $\mathcal{A} = \{1, 2, 3, 4, 5, 6, 7, 8\}$, the initial background $\mathcal{B} = \emptyset$, and that there are two minimal p-sets wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$, $X = \{3, 4, 7\}$ and $Y = \{4, 5, 8\}$. Further, suppose that QX pursues a splitting strategy where a set is always partitioned into equal-sized subsets in each iteration, i.e., SPLIT(n) returns $\lceil \frac{n}{2} \rceil$ (note: this leads to the best worst-case complexity of QX, cf. [8]).

Notation: Below, we show the workings of QX on this example by means of a tried and tested "flat" notation.[14] In this notation, the single-underlined subset denotes the current input to the function p in line 9, the double-underlined elements are those that are already fixed elements of the returned minimal p-set, and the grayed out elements those that are definitely not in the returned minimal p-set. Finally, ① signifies that the tested set (single-underlined along with double-underlined elements) is a p-set (function p in line 9 returns 1); ⓪ means it is no p-set (function p in line 9 returns 0).[15]

How QX *Proceeds:* After verifying that there is a non-empty p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$ and that $|\mathcal{A}| > 1$ (i.e., after the checks in lines 2 and 4 are negative, QX′ is called in line 7, and the checks in the first execution of lines 9 and 11 are negative), QX performs the following actions:

(1)	$[\underline{1, 2, 3, 4}, 5, 6, 7, 8]$	⓪	→	some element of p-set among 5,6,7,8
(2)	$[\underline{1, 2, 3, 4, 5, 6}, 7, 8]$	⓪	→	some element of p-set among 7,8
(3)	$[\underline{1, 2, 3, 4, 5, 6, 7}, 8]$	①	→	7 found, 8 irrelevant
(4)	$[\underline{1, 2, 3, 4}, 5, 6, \underline{7}, 8]$	①	→	5,6 irrelevant
(5)	$[1, 2, 3, 4, 5, 6, \underline{7}, 8]$	⓪	→	some element of p-set among 1,2,3,4
(6)	$[\underline{1, 2}, 3, 4, 5, 6, \underline{7}, 8]$	⓪	→	some element of p-set among 3,4
(7)	$[\underline{1, 2, 3}, 4, 5, 6, \underline{7}, 8]$	⓪	→	4 found
(8)	$[\underline{1, 2}, 3, \underline{4}, 5, 6, \underline{7}, 8]$	⓪	→	3 found
(9)	$[1, 2, \underline{\underline{3, 4}}, 5, 6, \underline{7}, 8]$	①	→	1,2 irrelevant

Explanation: After splitting $\mathcal{A}$ into two subsets of equal size, in step (1), QX

[14] We intentionally abstain from a notation which is guided by the call-recursion-tree or which lists all variables and their values (which we found was often perceived difficult to understand, e.g., since same variable names are differently assigned in all the recursive calls). The reason is: While explaining QX to people (mostly computer scientists) using various representations, we found out via people's feedback that the presented "flat" notation could best convey the intuition behind QX; moreover, it enabled people to correctly solve new examples on their own.

[15] An example of a different notation that describes the workings of QX based on the call-recursion-tree can be found, e.g., in [4, Fig. 4.1].

tests if there is a p-set in the left half $\{1,2,3,4\}$. Since negative, the right half $\{5,6,7,8\}$ is again split into equal-sized subsets, and the left one $\{5,6\}$ is added to the left half $\{1,2,3,4\}$ of the original set. Because this larger set $\{1,2,3,4,5,6\}$ still does not contain any p-set, the right subset $\{7,8\}$ is again split and the left part (7) added to the tested set, yielding $\{1,2,3,4,5,6,7\}$. Due to the positive predicate-test for this set, 7 is confirmed as an element of the found minimal p-set, and 8 is irrelevant. From now on, 7, as a fixed element of the p-set, takes part in all further executed predicate tests.

In step (4), the goal is to figure out whether the left half $\{5,6\}$ of $\{5,6,7,8\}$ also contains relevant elements. To this end, the left half $\{1,2,3,4\}$ of $\mathcal{A}$, along with 7, is tested, and positive. Therefore, a p-set is included in $\{1,2,3,4,7\}$ and $\{5,6\}$ is irrelevant. At this point, the output of the left subtree of the root, the one that analyzed $\{5,6,7,8\}$, is determined and fixed, i.e., is given by 7. The next task is to find the relevant elements in the right subtree, i.e., among $\{1,2,3,4\}$. As a consequence, in step (5), 7 alone is tested to check if all elements of $\{1,2,3,4\}$ are irrelevant. The result is negative, which is why the left half is split, and the left subset $\{1,2\}$ is tested along with 7, also negative. Thus, $\{3,4\}$ does include relevant elements. In step (7), QX finds that the element 3 alone from the set $\{3,4\}$ does not suffice to produce a p-set, i.e., the test for $\{1,2,3,7\}$ is negative. This lets us conclude that 4 must be in the p-set. So, 4 is fixed. To check the relevance of 3, $\{1,2,4,7\}$ is tested, yielding a negative result, which proves that 3 is relevant. The final test in step (9) if $\{1,2\}$ includes relevant elements as well, is negative, and 1,2 marked irrelevant. The set $\{3,4,7\}$ is finally returned, which coincides with X, one of our minimal p-sets. □

5. Proof of QX

In this section, we give a formal proof of the termination and soundness of the QX algorithm depicted by Alg. 1. By "soundness" we refer to the property that QX outputs a minimal p-set wrt. the p-PI it gets as an input, if a p-set exists, and 'no p-set' otherwise. While reading and thinking through the proof, the reader might consider it insightful to keep track of the meaning, implications, and interrelations of the various propositions in the proof by means of Fig. 1.[16]

Proposition 2 (Termination)**.** *Let $\langle\mathcal{A},\mathcal{B}\rangle$ be a p-PI. Then* $\mathrm{QX}(\langle\mathcal{A},\mathcal{B}\rangle)$ *terminates.*[17]

Proof. First, observe that QX either reaches line 7 (where QX' is called) or terminates before (in line 3 or line 5). Hence, $\mathrm{QX}(\langle\mathcal{A},\mathcal{B}\rangle)$ always terminates

[16] In the proof, we will often talk about different calls of $\mathrm{QX}'(\Delta,\langle\mathcal{A},\mathcal{B}\rangle)$ while $\mathrm{QX}(\langle\mathcal{A},\mathcal{B}\rangle)$ executes. To account for the facts that *(a)* the actual parameters passed to QX' will generally differ at each call, and *(b)* the actual parameters passed to QX' will generally not be equal to the original $\mathcal{A},\mathcal{B}$ (passed to QX), we (have to) use different designators $X', \bar{X}, \dot{X}$ and $\ddot{X}$ for these parameters $X \in \{\Delta,\mathcal{A},\mathcal{B}\}$ for each discussed QX'-call.

[17] Cf. Theorem 1 (unproven) in [8].

iff $\mathrm{QX}'(\mathcal{B}, \langle \mathcal{A}, \mathcal{B} \rangle)$ always terminates. We next show that $\mathrm{QX}'(\mathcal{B}, \langle \mathcal{A}, \mathcal{B} \rangle)$ terminates for an arbitrary p-PI $\langle \mathcal{A}, \mathcal{B} \rangle$.

$\mathrm{QX}'(\mathcal{B}, \langle \mathcal{A}, \mathcal{B} \rangle)$ either terminates directly (in that it returns in line 10 or line 12) or calls itself recursively in lines 16 and 17. However, for each recursive call $\mathrm{QX}'(\Delta', \langle \mathcal{A}', \mathcal{B}' \rangle)$ within $\mathrm{QX}'(\mathcal{B}, \langle \mathcal{A}, \mathcal{B} \rangle)$ it holds that $\emptyset \subset \mathcal{A}' \subset \mathcal{A}$ as $\mathcal{A}' \in \{\mathcal{A}_1, \mathcal{A}_2\}$ (see lines 14 and 15) and $\emptyset \subset \mathcal{A}_1, \mathcal{A}_2 \subset \mathcal{A}$ due to the definition of the SPLIT and GET functions.

Now, assume an infinite sequence of nested recursive calls of QX'. Since $\mathcal{A}$ is finite (Def. 2), this means that there must be a call $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ in this sequence where $|\bar{\mathcal{A}}| = 1$ and lines 16 and 17 (next nested recursive call in the infinite sequence) are reached. This is a contradiction to the fact that the test in line 11 enforces a return in line 12 given that $|\bar{\mathcal{A}}| = 1$. Consequently, every sequence of nested recursive calls during the execution of $\mathrm{QX}'(\mathcal{B}, \langle \mathcal{A}, \mathcal{B} \rangle)$ is finite (i.e., the depth of the call tree is finite).

Finally, there can only be a finite number of such nested recursive call sequences because no more than two recursive calls are made in any execution of QX' (i.e., the branching factor of the call tree is 2). This completes the proof. □

The following proposition witnesses that QX is sound in case the sub-procedure QX' is never called.

Proposition 3 (Correctness of QX When Trivial Cases Apply)**.**

(1) $\mathrm{QX}(\langle \mathcal{A}, \mathcal{B} \rangle)$ *returns 'no p-set' in line 3 iff there is no p-set wrt.* $\langle \mathcal{A}, \mathcal{B} \rangle$*.*
(2) If $\mathrm{QX}(\langle \mathcal{A}, \mathcal{B} \rangle)$ *returns* $\emptyset$ *in line 5,* $\emptyset$ *is a minimal p-set wrt.* $\langle \mathcal{A}, \mathcal{B} \rangle$*.*
(3) If the execution of $\mathrm{QX}(\langle \mathcal{A}, \mathcal{B} \rangle)$ *reaches line 7,* $p(\mathcal{A} \cup \mathcal{B}) = 1$ *holds.*

Proof. We prove all statements (1)–(3) in turn.

Proof of (1): The fact follows directly from Prop. 1.(1) and the test performed in line 2.

Proof of (2): Because line 5 is reached, $p(\mathcal{A} \cup \mathcal{B}) = 1$ (as otherwise a return would have taken place at line 3) and $\mathcal{A} = \emptyset$ (due to line 4) must hold. Since $p(\mathcal{A} \cup \mathcal{B}) = 1$ implies the existence of a p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$ by Prop. 1.(1), and since any p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$ must be a subset of $\mathcal{A}$ by Def. 3, $\emptyset$ is the only (and therefore trivially a minimal) p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$.

Proof of (3): This statement follows directly from the test in line 2 and the fact that line 7 is reached. □

We now characterize an invariant which applies to every call of QX' throughout the execution of QX.

Definition 4 (Invariant Property of QX')**.** *Let* $\mathrm{QX}'(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$ *be a call of* QX'*. Then we say that* $\mathsf{Invar}(\Delta, \mathcal{A}, \mathcal{B})$ *holds for this call iff*

$$(\Delta \neq \emptyset \vee p(\mathcal{B}) = 0) \ \wedge \ p(\mathcal{A} \cup \mathcal{B}) = 1$$

The next proposition shows that this invariant holds for the first call of QX' in Alg. 1.

Proposition 4 (Invariant Holds For First Call of QX')**.** $\mathsf{Invar}(\bar{\Delta}, \bar{\mathcal{A}}, \bar{\mathcal{B}})$ *holds for* $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ *given that* $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ *was called in line 7.*

Proof. Since $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ was called in line 7, we have $\bar{\Delta} = \mathcal{B}$, $\bar{\mathcal{A}} = \mathcal{A}$ and $\bar{\mathcal{B}} = \mathcal{B}$. Since $p(\mathcal{A} \cup \mathcal{B}) = 1$ holds in line 7 on account of Prop. 3.(3), we have that $p(\bar{\mathcal{A}} \cup \bar{\mathcal{B}}) = 1$. To show that $(\bar{\Delta} \neq \emptyset \vee p(\bar{\mathcal{B}}) = 0)$, we distinguish the cases $\mathcal{B} = \emptyset$ and $\mathcal{B} \neq \emptyset$. Let first $\mathcal{B} = \emptyset$. Due to Def. 1, we have that $p(\bar{\mathcal{B}}) = p(\mathcal{B}) = p(\emptyset) = 0$. Second, assume $\mathcal{B} \neq \emptyset$. Since $\bar{\Delta} = \mathcal{B}$, we directly obtain that $\bar{\Delta} \neq \emptyset$. □

Given the invariant of Def. 4 holds for some call of QX', we next demonstrate that the output returned by QX' is sound (i.e., a minimal p-set) when it returns in line 10 or 12 (i.e., if this call of QX' represents a leaf node in the call-recursion-tree). Moreover, we show that the invariant is "propagated" to the recursive call of QX' in line 16 (i.e., this invariant remains valid as long as the algorithm keeps going downwards in the call-recursion-tree).

Proposition 5 (Invariant Causes Sound Outputs and Propagates Downwards)**.** *If* $\mathsf{Invar}(\Delta, \mathcal{A}, \mathcal{B})$ *holds for* $\mathrm{QX}'(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$*, then:*

(1) $\mathrm{QX}'(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$ *returns* $\emptyset$ *in line 10 iff* $\emptyset$ *is a (minimal) p-set wrt.* $\langle \mathcal{A}, \mathcal{B} \rangle$*.*
(2) If the execution of $\mathrm{QX}'(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$ *reaches line 11, then* $p(\mathcal{B}) = 0$ *holds.*
(3) If $\mathrm{QX}'(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$ *returns* $\mathcal{A}$ *in line 12, then* $\mathcal{A}$ *is a minimal p-set wrt.* $\langle \mathcal{A}, \mathcal{B} \rangle$*.*
(4) If the execution of $\mathrm{QX}'(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$ *reaches line 16, where* $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ *is called, then* $\mathsf{Invar}(\bar{\Delta}, \bar{\mathcal{A}}, \bar{\mathcal{B}})$*.*

Proof. We prove all statements (1)–(4) in turn.

Proof of (1): "⇒": We assume that $\mathrm{QX}'(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$ returns in line 10. By the test performed in line 9, this can only be the case if $p(\mathcal{B}) = 1$. By Prop. 1.(2), this implies that $\emptyset$ is a (minimal) p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$.
"⇐": We assume that $\emptyset$ is a (minimal) p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$. To show that a return takes place in line 10, we have to prove that the condition tested in line 9 is true. First, we observe that $p(\mathcal{B}) = 1$ must hold due to Prop. 1.(2). Since $\mathsf{Invar}(\Delta, \mathcal{A}, \mathcal{B})$ holds (see Def. 4), we can infer from $p(\mathcal{B}) = 1$ that $\Delta \neq \emptyset$. Hence, the condition in line 9 is satisfied.

Proof of (2): Prop. 5.(1) shows that line 11 is reached iff $\emptyset$ is not a p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$ which is the case iff $p(\mathcal{B}) = 0$ due to Prop. 1.(2).

Proof of (3): A return in line 12 can only occur if the test in line 11 is positive, i.e., if line 11 is reached and $|\mathcal{A}| = 1$. Moreover, since $\mathsf{Invar}(\Delta, \mathcal{A}, \mathcal{B})$ holds, it follows that $p(\mathcal{A} \cup \mathcal{B}) = 1$.

First, $p(\mathcal{A} \cup \mathcal{B}) = 1$ is equivalent to the existence of a p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$. Second, by Def. 3, a p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$ is a subset of $\mathcal{A}$. Third, $|\mathcal{A}| = 1$ means that $\emptyset$ and $\mathcal{A}$ are all possible subsets of $\mathcal{A}$. Fourth, since line 11 is reached, we have that $p(\mathcal{B}) = 0$ by statement (2) of this Proposition, which implies that $\emptyset$

is not a p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$ according to Prop. 1.(2). Consequently, $\mathcal{A}$ must be a minimal p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$.

Proof of (4): Consider the call $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ at line 16. Due to the definition of the SPLIT and GET functions ($1 \leq k \leq |\mathcal{A}| - 1$, $\mathcal{A}_1$ includes the first k, $\mathcal{A}_2$ the last $|\mathcal{A}| - k$ elements of $\mathcal{A}$) and the fact that $\bar{\Delta} = \mathcal{A}_1$, the property $\bar{\Delta} \neq \emptyset$ must hold. Moreover, $\bar{\mathcal{A}} \cup \bar{\mathcal{B}} = \mathcal{A}_2 \cup \mathcal{B} \cup \mathcal{A}_1 = \mathcal{A} \cup \mathcal{B}$. Due to $\mathsf{Invar}(\Delta, \mathcal{A}, \mathcal{B})$, however, we know that $p(\mathcal{A} \cup \mathcal{B}) = 1$. Therefore, $p(\bar{\mathcal{A}} \cup \bar{\mathcal{B}}) = 1$ must be true. According to Def. 4, it follows that $\mathsf{Invar}(\bar{\Delta}, \bar{\mathcal{A}}, \bar{\mathcal{B}})$ holds. □

Note, immediately before line 17 is first reached during the execution of QX, it must be the case that, for the first time, a recursive call $\mathrm{QX}'(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$ made in line 16 returns (i.e., we reach a leaf node in the call-recursion-tree for the first time and the first "backtracking" takes place). By Prop. 5.(1)+(3), the output of this call $\mathrm{QX}'(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$, namely X_2 in line 16, is a minimal p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$. We now prove that the invariant property given in Def. 4 in this case "propagates" to the first-ever call of QX' in line 17.

Proposition 6 (If Output of Left Sub-Tree is Sound, Invariant Propagates to Right Sub-Tree)**.** *Let* $\mathsf{Invar}(\Delta, \mathcal{A}, \mathcal{B})$ *be true for some call* $\mathrm{QX}'(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$ *and let the recursive call* $\mathrm{QX}'(\dot{\Delta}, \langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle)$ *in line 16 during the execution of* $\mathrm{QX}'(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$ *return a minimal p-set wrt.* $\langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle$*. Then* $\mathsf{Invar}(\ddot{\Delta}, \ddot{\mathcal{A}}, \ddot{\mathcal{B}})$ *holds for the recursive call* $\mathrm{QX}'(\ddot{\Delta}, \langle \ddot{\mathcal{A}}, \ddot{\mathcal{B}} \rangle)$ *in line 17 during the execution of* $\mathrm{QX}'(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$.

Proof. As per Def. 4, we have to show that $(\ddot{\Delta} \neq \emptyset \vee p(\ddot{\mathcal{B}}) = 0) \wedge p(\ddot{\mathcal{A}} \cup \ddot{\mathcal{B}}) = 1$.

We first prove $p(\ddot{\mathcal{A}} \cup \ddot{\mathcal{B}}) = 1$. Since X_2, the set returned by $\mathrm{QX}'(\dot{\Delta}, \langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle) = \mathrm{QX}'(\mathcal{A}_1, \langle \mathcal{A}_2, \mathcal{B} \cup \mathcal{A}_1 \rangle)$ in line 16, is a minimal p-set wrt. $\langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle = \langle \mathcal{A}_2, \mathcal{B} \cup \mathcal{A}_1 \rangle$, we infer by Def. 3 that $p(X_2 \cup \mathcal{B} \cup \mathcal{A}_1) = 1$. However, it holds that $\mathrm{QX}'(\ddot{\Delta}, \langle \ddot{\mathcal{A}}, \ddot{\mathcal{B}} \rangle) = \mathrm{QX}'(X_2, \langle \mathcal{A}_1, \mathcal{B} \cup X_2 \rangle)$. Therefore, $p(\ddot{\mathcal{A}} \cup \ddot{\mathcal{B}}) = p([\mathcal{A}_1] \cup [\mathcal{B} \cup X_2]) = 1$.

It remains to be shown that $(\ddot{\Delta} \neq \emptyset \vee p(\ddot{\mathcal{B}}) = 0)$ holds, which is equivalent to $(X_2 \neq \emptyset \vee p(\mathcal{B} \cup X_2) = 0)$. If $X_2 \neq \emptyset$, we are done. So, let us assume that $X_2 = \emptyset$. In this case, however, we have $p(\mathcal{B} \cup X_2) = p(\mathcal{B})$. As $\mathsf{Invar}(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$ holds and line 17 is reached during the execution of $\mathrm{QX}'(\Delta, \langle \mathcal{A}, \mathcal{B} \rangle)$, we know by Prop. 5.(2) that $p(\mathcal{B}) = 0$. Hence, $p(\mathcal{B} \cup X_2) = p(\mathcal{B}) = 0$.

Overall, we have demonstrated that $\mathsf{Invar}(\ddot{\Delta}, \langle \ddot{\mathcal{A}}, \ddot{\mathcal{B}} \rangle)$ holds. □

At this point, we know that the invariant property of Def. 4 remains valid up to and including the first recursive call of QX' in line 17 (i.e., until immediately after the first leaf in the call-recursion-tree is encountered, a single-step "backtrack" is made, and the first branching to the right is executed). From then on, as long as only "downward" calls of QX' in line 16, possibly interleaved with single calls of QX' in line 17, are performed, the validity of the invariant is preserved.

Due to the fact that QX terminates (Prop. 2), the call-recursion-tree must be finite. Hence, the situation must occur, where QX' called in line 16 directly

returns (i.e., in line 10 or 12) and the immediately subsequent call of QX' in line 17 directly returns (i.e., in line 10 or 12) as well (i.e., we face the situation where both the left and the right branch at one node in the call-recursion-tree consist only of a single leaf node). As the invariant holds in this right branch, the said call of QX' in line 17 must indeed return a minimal p-set wrt. its p-PI given as an argument, due to Prop. 5.(1)+(3).

The next proposition evidences—as a special case—that the combination (set-union) of the two outputs X_2 (left leaf node) and X_1 (right leaf node) returned in line 18 in fact constitutes a minimal p-set for the p-PI given as an input argument to the call of QX' which executes line 18. More generally, the proposition testifies that, given the calls in line 16 and line 17 each return a minimal p-set wrt. their given p-PIs—whether or not these calls directly return—the combination of these p-sets is again a minimal p-set for the respective p-PI at the call that executed lines 16 and 17.

Proposition 7 (If Output of Both Left and Right Sub-Tree is Sound, then a Sound Result is Returned (Propagated Upwards))**.** *Let the recursive call* $\mathrm{QX}'(\dot{\Delta}, \langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle)$ *in line 16 during the execution of* $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ *return a minimal p-set wrt.* $\langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle$*, and let the recursive call* $\mathrm{QX}'(\ddot{\Delta}, \langle \ddot{\mathcal{A}}, \ddot{\mathcal{B}} \rangle)$ *in line 17 during the execution of* $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ *return a minimal p-set wrt.* $\langle \ddot{\mathcal{A}}, \ddot{\mathcal{B}} \rangle$*. Then* $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ *returns a minimal p-set wrt.* $\langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle$*.*

Proof. The statement is a direct consequence of Lemma 1 below. □

Lemma 1. *Let* $\mathcal{A}_1, \mathcal{A}_2$ *be a partition of* $\mathcal{A}$*. If* (a) X_2 *is a minimal p-set wrt.* $\langle \mathcal{A}_2, \mathcal{B} \cup \mathcal{A}_1 \rangle$ *and* (b) X_1 *is a minimal p-set wrt.* $\langle \mathcal{A}_1, \mathcal{B} \cup X_2 \rangle$*, then* $X_1 \cup X_2$ *is a minimal p-set wrt.* $\langle \mathcal{A}, \mathcal{B} \rangle$*.*[18]

Proof. We first show that $X_1 \cup X_2$ is a p-set, and then we show its minimality.

p-set property: First, by Def. 3, $X_1 \subseteq \mathcal{A}_1$ due to (a), and $X_2 \subseteq \mathcal{A}_2$ due to (b), which is why $X_1 \cup X_2 \subseteq \mathcal{A}_1 \cup \mathcal{A}_2 = \mathcal{A}$. From the fact that X_1 is a minimal p-set wrt. $\langle \mathcal{A}_1, \mathcal{B} \cup X_2 \rangle$, along with Def. 3, we get $p(X_1 \cup [\mathcal{B} \cup X_2]) = 1 = p([X_1 \cup X_2] \cup \mathcal{B})$. Hence, $X_1 \cup X_2$ is a p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$ due to Def. 3.

Minimality: To show that $X_1 \cup X_2$ is a *minimal* p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$, assume that $X \subset X_1 \cup X_2$ is a p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$. The set X can be represented as $X = X'_1 \cup X'_2$ where (1) $X'_1 := X \cap X_1 \subseteq X_1$ and (2) $X'_2 := X \cap X_2 \subseteq X_2$. In addition, the $\subseteq$-relation in (1) or (2) must be a $\subset$-relation, i.e., X does not include all elements of X_1 or not all elements of X_2.

Let us first assume that $\subset$ holds in (1). Then, $X = X'_1 \cup X'_2$ where $X'_1 \subset X_1$ and $X'_2 \subseteq X_2$. Since X is a p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$, we have $p(X \cup \mathcal{B}) = p([X'_1 \cup X'_2] \cup \mathcal{B}) = p(X'_1 \cup [\mathcal{B} \cup X'_2]) = 1$. By monotonicity of p, it follows that $p(X'_1 \cup [\mathcal{B} \cup X_2]) = 1$. Because of $X'_1 \subset X_1 \subseteq \mathcal{A}_1$, we have that X'_1 is a p-set wrt. $\langle \mathcal{A}_1, \mathcal{B} \cup X_2 \rangle$, which is a contradiction to the premise (b).

[18] Cf. Prop. 6.2 (unproven) in [8].

Second, assume that $\subset$ holds in (2). Then, $X = X'_1 \cup X'_2$ where $X'_1 \subseteq X_1$ and $X'_2 \subset X_2$. Since X is a p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$, we have $p(X \cup \mathcal{B}) = p([X'_1 \cup X'_2] \cup \mathcal{B}) = p(X'_2 \cup [\mathcal{B} \cup X'_1]) = 1$. By monotonicity of p, and since $X'_1 \subseteq X_1 \subseteq \mathcal{A}_1$, it follows that $p(X'_2 \cup [\mathcal{B} \cup \mathcal{A}_1]) = 1$. As $X'_2 \subset X_2 \subseteq \mathcal{A}_2$, we obtain that X'_2 is a p-set wrt. $\langle \mathcal{A}_2, \mathcal{B} \cup \mathcal{A}_1 \rangle$, which is a contradiction to premise (a). □

Proposition 8 (If Invariant Holds for Tree, Then a Minimal p-Set is Returned By Tree)**.** *If* $\mathsf{Invar}(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ *holds for* $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$*, then it returns a minimal p-set wrt.* $\langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle$*.*

Proof. We prove this proposition by induction on d where d is the maximal number of *recursive*[19] calls of QX' on the call stack throughout the execution of $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$.

Induction Base: Let $d = 0$. That is, no recursive calls are executed, or, equivalently, $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ returns in line 10 or 12. Since $\mathsf{Invar}(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ is true, a minimal p-set wrt. $\langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle$ is returned, which follows from Prop. 5.(1)+(3).

Induction Assumption: Let the statement of the proposition be true for $d = k$. We will now show that, in this case, the statement holds for $d = k + 1$ as well.

Induction Step: Assume that (at most) $k + 1$ recursive calls are ever on the call stack while $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ executes. Since $\mathsf{Invar}(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ holds, Prop. 5.(4) lets us conclude that $\mathsf{Invar}(\dot{\Delta}, \langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle)$ holds for the first recursive call $\mathrm{QX}'(\dot{\Delta}, \langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle)$ issued in line 16 of $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$. Now, we have that, for $\mathrm{QX}'(\dot{\Delta}, \langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle)$, the maximal number of recursive calls ever on the call stack while it executes, is (at most) k. Therefore, by the *Induction Assumption*, $\mathrm{QX}'(\dot{\Delta}, \langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle)$ returns a minimal p-set wrt. $\langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle$.

Because $\mathsf{Invar}(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ holds and $\mathrm{QX}'(\dot{\Delta}, \langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle)$ called in line 16 during the execution of $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ returns a minimal p-set wrt. $\langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle$, we deduce by means of Prop. 6 that $\mathsf{Invar}(\ddot{\Delta}, \langle \ddot{\mathcal{A}}, \ddot{\mathcal{B}} \rangle)$ holds for the call $\mathrm{QX}'(\ddot{\Delta}, \langle \ddot{\mathcal{A}}, \ddot{\mathcal{B}} \rangle)$ made in line 17 during the execution of $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$. Again, it must be true that the maximal number of recursive calls ever on the call stack while $\mathrm{QX}'(\ddot{\Delta}, \langle \ddot{\mathcal{A}}, \ddot{\mathcal{B}} \rangle)$ executes is (at most) k. Consequently, $\mathrm{QX}'(\ddot{\Delta}, \langle \ddot{\mathcal{A}}, \ddot{\mathcal{B}} \rangle)$ returns a minimal p-set wrt. $\langle \ddot{\mathcal{A}}, \ddot{\mathcal{B}} \rangle$ due to the *Induction Assumption*.

As both recursive calls made throughout the execution of $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ return a minimal p-set wrt. their given p-PIs $\langle \dot{\mathcal{A}}, \dot{\mathcal{B}} \rangle$ and $\langle \ddot{\mathcal{A}}, \ddot{\mathcal{B}} \rangle$, respectively, we conclude by Prop. 7 that $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ returns a minimal p-set wrt. $\langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle$.

This completes the inductive proof. □

Theorem 1 (Correctness of QX)**.** *Let* $\langle \mathcal{A}, \mathcal{B} \rangle$ *be a p-PI. Then,* $\mathrm{QX}(\langle \mathcal{A}, \mathcal{B} \rangle)$ *returns a minimal p-PI wrt.* $\langle \mathcal{A}, \mathcal{B} \rangle$ *if a p-set exists for* $\langle \mathcal{A}, \mathcal{B} \rangle$*. Otherwise,* $\mathrm{QX}(\langle \mathcal{A}, \mathcal{B} \rangle)$ *returns 'no p-set'.*[20]

[19] That is, *additional* calls made, *not* taking into account the running routine $\mathrm{QX}'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ that we consider in the proposition.

[20] Cf. Theorem 1 in [8].

Proof. Prop. 3.(1), first, proves that 'no p-set' is returned if there is no p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$. Second, it shows that, if there is a p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$, QX will either return in line 5 or call QX$'$ in line 7.

We now show that, in both of these cases, QX returns a minimal p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$. This then implies that a minimal p-set is returned by QX whenever such a one exists.

First, if QX returns in line 5, then the output is a minimal p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$ due to Prop. 3.(2).

Second, if QX-calls QX$'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ in line 7, then $\mathsf{Invar}(\bar{\Delta}, \bar{\mathcal{A}}, \bar{\mathcal{B}})$ holds according to Prop. 4. Finally, since $\mathsf{Invar}(\bar{\Delta}, \bar{\mathcal{A}}, \bar{\mathcal{B}})$ holds for QX$'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$, Prop. 8 establishes that QX$'(\bar{\Delta}, \langle \bar{\mathcal{A}}, \bar{\mathcal{B}} \rangle)$ returns a minimal p-set wrt. $\langle \mathcal{A}, \mathcal{B} \rangle$. □

6. General Advice on How to Prove a Recursive Algorithm

Since we did not find any literature addressing the topic of how to principally approach the proof of a recursive algorithm, we briefly outline a general template (which we also stuck to in our proof) that can serve as a guideline when intending to show the correctness of a recursive procedure. The foundation for this template is provided by the proof principle for (non-recursive) algorithms based on loop invariants detailed by Cormen et al. in [52]. The idea underlying their framework (which we shall call L-INV) is to prove that a *loop invariant*, i.e., a predicate that is always true while a loop executes, *(1)* holds when the loop is entered (*L-initialization*), *(2)* remains true for the next loop iteration if it is true for the current iteration (*L-maintenance*), and *(3)* yields a property at termination of the loop that is useful to prove the algorithm's correctness (*L-termination*).[21] The template to prove the correctness of recursive algorithms (dubbed R-INV) we propose

- relies on a *recursion invariant*, i.e., a predicate that is true for every recursive call of a procedure,
- involves the proof that this invariant
  - holds for the first call of the recursive procedure (*R-initialization*),
  - remains true for any further (recursive) call of the procedure (*R-maintenance*),
  - entails correctness of the procedure's output (*R-termination*).

In spite of the resemblance between R-INV and L-INV, it is essential to understand the different nature of the proof when considering a recursive as opposed to a non-recursive procedure. Whereas R-initialization, similarly as L-initialization, will often be quite easily shown due to its independence from the recursion, addressing R-maintenance and R-termination is more elaborate than L-maintenance and L-termination in general. The reasons are as follows:

[21] The three stages of the proof are originally called initialization, maintenance and termination in [52]; we added the L-prefix to distinguish these terms relating to loops from the newly introduced ones, which are associated with recursion (R-prefix).

1. There is only one entry point into a loop, whereas there may be multiple different places where a recursive procedure can be called. *Consequence:* There is one case to be analyzed for L-maintenance, whereas there can be *multiple cases to be distinguished* for R-maintenance.
2. For a loop, there is no return value and often[22] only one termination condition, whereas there are multiple termination conditions[23] for a recursion, and for each such condition a different value can be returned. In particular, a recursive procedure can return due to some trivial case that applies (no nested recursive calls) or after recursively processing a non-trivial case (nested recursive calls). *Consequence:* There is usually one case to be analyzed for L-termination, whereas *multiple cases need to be considered* for R-termination. Moreover, demonstrating R-termination when a non-trivial case is processed by the recursive procedure *requires an induction proof.*
3. In case the recursion implements a divide-and-conquer approach, there can be *combine*-steps where partial solutions are integrated to a complete solution (cf. Alg. 1, line 18). *Consequence:* These *combine-steps need to be addressed* when proving R-termination.

As an illustration, the following table shows how the building blocks of our proof are assigned to the three different proof phases of R-INV:[24]

R-initialization	(abbreviations: P...Proposition, T...Theorem)
P4 (invariant holds for first call of QX′)	
R-maintenance	
(case 1: recursive QX′*-call 1)* P5.4 (invariant ⇒ invariant holds for recursive call in line 16)	
(case 2: recursive QX′*-call 2)* P6 (invariant ⇒ invariant holds for recursive call in line 17)	
R-termination	
(correctness of combine-step) P7 (correct outputs in lines 16, 17 ⇒ correct output in line 18)	
(trivial case) P5.1 + P5.3 (invariant ⇒ correct output if no nested recursive calls)	
(general case: proof by induction) P8 (invariant ⇒ correct output given nested recursive calls)	

Finally, we remark that finding the "right" invariant can be tricky and will often represent the key to success in proving a recursive algorithm by means of the R-INV template. Also in case of the presented proof (cf. Sec. 5), after having determined the appropriate invariant, the rest of the proof is relatively straightforward when following the systematic steps suggested by R-INV.

7. Conclusion

QuickXPlain (QX) is a very popular, highly cited, and frequently employed, adapted, and extended algorithm to solve the MSMP problem, i.e., to

[22] If break-statements do not occur within the loop.

[23] One such "condition" is reaching the last statement of the procedure.

[24] To be precise, the table shows the steps of the proof that QX′, i.e., the recursive part of QX, is correct. To show that QX is correct, Theorem 1 finally simply combines the correctness of the trivial cases (checked before QX′ is called) with the correctness of QX′.

find a subset of a given universe such that this subset is irreducible subject to a monotone predicate (e.g., logical consistency). MSMP is an important and common problem and its manifestations occur in a wide range of computer science disciplines. Since QX has in practice turned out to be hardly understood by many—experienced academics included—and was published without a proof, we account for that by providing for QX an intelligible *proof that explains*. The availability and accessibility of a formal proof is instrumental in various regards. Beside allowing the verification of QX's correctness (*proof effect*), it fosters proper and full understanding of QX and of other works relying on QX (*didactic effect*), it is a necessary foundation for "gapless" correctness proofs of numerous algorithms, e.g., in model-based diagnosis, that rely on (results computed by) QX (*completeness effect*), it makes the intuition of QX's correctness bullet-proof and excludes the later detection of algorithmic errors, as was already experienced even for seminal works in the past (*trust and sustainability effect*), as well as it might be used as a template for devising proofs of other recursive algorithms (*transfer effect*). Since *(i)* we exemplify the workings of QX using a novel tried and tested well-comprehensible notation, and *(ii)* we put a special emphasis on the clarity and didactic value of the given proof (e.g., by segmenting the proof into small, intuitive, and easily-digestible chunks, by showing how our proof can be "directly traced" using the recursive call tree produced by QX, and by explaining the underlying proof template with the intention to make it reusable for other proofs), we believe that this work can decisively contribute to a better understanding of QX, which we expect to be of great value for both practitioners and researchers.

Acknowledgments

This work was supported by the Austrian Science Fund (FWF), contract P-32445-N38. Thanks to Dietmar Jannach and to the anonymous referees for valuable comments that helped improve this manuscript.

References

[1] A. R. Bradley, Z. Manna, Checking safety by inductive generalization of counterexamples to induction, in: Formal Methods in Computer Aided Design, 2007.

[2] J. Marques-Silva, M. Janota, A. Belov, Minimal sets over monotone predicates in boolean formulae, in: International Conference on Computer Aided Verification, 2013.

[3] D. Jannach, T. Schmitz, Model-based diagnosis of spreadsheet programs: A constraint-based debugging approach, Automated Software Engineering 23 (1) (2016) 105–144.

[4] P. Rodler, Interactive Debugging of Knowledge Bases, Ph.D. thesis, Univ. Klagenfurt, CoRR abs/1605.05950 (2015).

[5] A. Kalyanpur, Debugging and Repair of OWL Ontologies, Ph.D. thesis, Univ. Maryland, College Park (2006).

[6] P. Rodler, M. Herold, StaticHS: A variant of Reiter's hitting set tree for efficient sequential diagnosis, in: 11th Annual Symposium on Combinatorial Search, 2018.

[7] U. Junker, QuickXPlain: Conflict Detection for Arbitrary Constraint Propagation Algorithms, in: IJCAI'01 Workshop on Modelling and Solving problems with Constraints, 2001.

[8] U. Junker, QuickXPlain: Preferred Explanations and Relaxations for Over-Constrained Problems, in: AAAI'04, 2004.

[9] C. Lecoutre, L. Sais, S. Tabary, V. Vidal, Recording and minimizing nogoods from restarts, Journal on Satisfiability, Boolean Modeling and Computation 1 (3-4) (2006) 147–167.

[10] A. R. Bradley, Z. Manna, Property-directed incremental invariant generation, Formal Aspects of Computing 20 (4-5) (2008) 379–405.

[11] A. Nadel, Boosting minimal unsatisfiable core extraction, in: Proceedings of the 2010 Conference on Formal Methods in Computer-Aided Design, 2010.

[12] Z. S. Andraus, M. H. Liffiton, K. A. Sakallah, Reveal: A formal verification tool for verilog designs, in: International Conference on Logic for Programming Artificial Intelligence and Reasoning, 2008.

[13] A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner, Consistency-based diagnosis of configuration knowledge bases, Artificial Intelligence 152 (2) (2004) 213–234.

[14] J. White, D. Benavides, D. C. Schmidt, P. Trinidad, B. Dougherty, A. Ruiz-Cortes, Automated diagnosis of feature model configurations, Journal of Systems and Software 83 (7) (2010) 1094–1107.

[15] A. Darwiche, Decomposable negation normal form, Journal of the ACM 48 (4) (2001) 608–647.

[16] J. McCarthy, Circumscription—A form of non-monotonic reasoning, Artificial intelligence 13 (1-2) (1980) 27–39.

[17] T. Eiter, G. Ianni, T. Krennwallner, Answer set programming: A primer, in: Reasoning Web International Summer School, 2009.

[18] P. Marquis, Knowledge compilation using theory prime implicates, in: IJCAI'95, 1995.

[19] A. Felfernig, G. Friedrich, D. Jannach, M. Zanker, An integrated environment for the development of knowledge-based recommender applications, International Journal of Electronic Commerce 11 (2) (2006) 11–34.

[20] A. Felfernig, M. Mairitsch, M. Mandl, M. Schubert, E. Teppan, Utility-based repair of inconsistent requirements, in: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, 2009.

[21] P. Rodler, K. Shchekotykhin, P. Fleiss, G. Friedrich, RIO: Minimizing User Interaction in Ontology Debugging, in: Web Reasoning and Rule Systems, 2013.

[22] C. Meilicke, Alignment Incoherence in Ontology Matching, Ph.D. thesis, Univ. Mannheim (2011).

[23] P. Rodler, D. Jannach, K. Schekotihin, P. Fleiss, Are query-based ontology debuggers really helping knowledge engineers?, Knowledge-Based Systems 179 (2019) 92–107.

[24] K. Shchekotykhin, G. Friedrich, P. Fleiss, P. Rodler, Interactive Ontology Debugging: Two Query Strategies for Efficient Fault Localization, Web Semantics: Science, Services and Agents on the World Wide Web 12-13 (2012) 88–103.

[25] M. Horridge, Justification based explanation in ontologies, Ph.D. thesis, Univ. Manchester (2011).

[26] S. Schlobach, Z. Huang, R. Cornet, F. Van Harmelen, Debugging incoherent terminologies, Journal of Automated Reasoning 39 (3) (2007) 317–349.

[27] K. Schekotihin, P. Rodler, W. Schmid, OntoDebug: Interactive ontology debugging plug-in for Protégé, in: International Symposium on Foundations of Information and Knowledge Systems, 2018.

[28] N. Dershowitz, Z. Hanna, A. Nadel, A scalable algorithm for minimal unsatisfiable core extraction, in: International Conference on Theory and Applications of Satisfiability Testing, 2006.

[29] Y. Oh, M. N. Mneimneh, Z. S. Andraus, K. A. Sakallah, I. L. Markov, Amuse: A minimally-unsatisfiable subformula extractor, in: Proceedings of the 41st annual Design Automation Conference, 2004.

[30] M. H. Liffiton, K. A. Sakallah, Algorithms for computing minimal unsatisfiable subsets of constraints, Journal of Automated Reasoning 40 (1) (2008) 1–33.

[31] R. Reiter, A Theory of Diagnosis from First Principles, Artificial Intelligence 32 (1) (1987) 57–95.

[32] J. de Kleer, B. C. Williams, Diagnosing multiple faults, Artificial Intelligence 32 (1) (1987) 97–130.

[33] E. Birnbaum, E. L. Lozinskii, Consistent subsets of inconsistent systems: Structure and behaviour, Journal of Experimental & Theoretical Artificial Intelligence 15 (1) (2003) 25–46.

[34] J. Marques-Silva, F. Heras, M. Janota, A. Previti, A. Belov, On computing minimal correction subsets, in: IJCAI'13, 2013.

[35] P. Rodler, Reuse, Reduce and Recycle: Optimizing Reiter's HS-Tree for Sequential Diagnosis, in: ECAI'20, 2020.

[36] J. R. Slagle, C.-L. Chang, R. C. Lee, A new algorithm for generating prime implicants, IEEE Transactions on Computers 100 (4) (1970) 304–310.

[37] W. V. Quine, On cores and prime implicants of truth functions, The American Mathematical Monthly 66 (9) (1959) 755–760.

[38] V. M. Manquinho, P. F. Flores, J. P. M. Silva, A. L. Oliveira, Prime implicant computation using satisfiability algorithms, in: Proceedings 9th IEEE International Conference on Tools with Artificial Intelligence, 1997.

[39] D. Déharbe, P. Fontaine, D. Le Berre, B. Mazure, Computing prime implicants, in: 2013 Formal Methods in Computer-Aided Design, 2013.

[40] P. Rodler, Towards better response times and higher-quality queries in interactive knowledge base debugging, Tech. rep., Univ. Klagenfurt, CoRR abs/1609.02584 (2016).

[41] P. Rodler, W. Schmid, K. Schekotihin, A generally applicable, highly scalable measurement computation and optimization approach to sequential model-based diagnosis, CoRR abs/1711.05508 (2017).

[42] K. Shchekotykhin, G. Friedrich, P. Rodler, P. Fleiss, Sequential diagnosis of high cardinality faults in knowledge-bases by direct diagnosis generation, in: ECAI'14, 2014.

[43] K. Shchekotykhin, D. Jannach, T. Schmitz, MergeXplain: Fast computation of multiple conflicts for diagnosis, in: IJCAI'15, 2015.

[44] A. Felfernig, M. Schubert, C. Zehentner, An efficient diagnosis algorithm for inconsistent constraint sets, AI EDAM 26 (1) (2012) 53–62.

[45] A. Belov, J. Marques-Silva, MUSer2: An efficient MUS extractor, Journal on Satisfiability, Boolean Modeling and Computation 8 (3-4) (2012) 123–128.

[46] J. Marques-Silva, I. Lynce, On improving MUS extraction algorithms, in: International Conference on Theory and Applications of Satisfiability Testing, 2011.

[47] K. Shchekotykhin, G. Friedrich, D. Jannach, On computing minimal conflicts for ontology debugging, Model-Based Systems 7.

[48] G. Hanna, H. N. Jahnke, Proof and proving, in: International Handbook of Mathematics Education, 1996.

[49] R. Greiner, B. A. Smith, R. W. Wilkerson, A correction to the algorithm in Reiter's theory of diagnosis, Artificial Intelligence 41 (1) (1989) 79–88.

[50] G. Hanna, Proof and its classroom role: A survey, Atas do Encontro de Investigação em Educação Matemática-IX EIEM (2000) 75–104.

[51] G. Hanna, Some pedagogical aspects of proof, Interchange 21 (1) (1990) 6–13.

[52] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to algorithms, MIT Press (2009).