The QSMA Algorithm for Quantifiers in SMT

Maria Paola Bonacina[1*], Stéphane Graham-Lengrand[2] and Christophe Vauthier[3]

[1*]Dipartimento di Informatica, Università degli Studi di Verona, Strada Le Grazie 15, Verona, 37134, VR, Italy.
[2]Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, 94025, CA, USA.
[3]Laboratoire de Mathématiques d'Orsay, Université Paris-Saclay, 3 Rue Joliot Curie, Gif-sur-Yvette, 91190, France.

*Corresponding author(s). E-mail(s): mariapaola.bonacina@univr.it;
Contributing authors: stephane.graham-lengrand@csl.sri.com; christophe.vauthier@universite-paris-saclay.fr;

Abstract

Deciding the satisfiability of formulas involving both quantifiers and theory defined symbols is a challenge in automated reasoning. This article presents an algorithm, called **QSMA** (*Quantified Satisfiability Modulo Assignment*), for the satisfiability of an arbitrary quantified formula modulo a complete theory and an initial assignment. The algorithm is proved partially correct and terminating, so that its total correctness is established. An optimized variant called **OptiQSMA** is also described and shown to preserve both partial correctness and termination. **OptiQSMA** is implemented in the YicesQS solver. **OptiQSMA** enabled YicesQS to achieve top of the line results, especially in linear rational arithmetic, in the 2022, 2023, and 2024 editions of the International Satisfiability Modulo Theories Competition (SMT-COMP). A report on these results in four fragments of arithmetic (**LRA** – Linear Rational Arithmetic, **LIA** – Linear Integer Arithmetic, **NRA** – Nonlinear Real Arithmetic, and **NIA** – Nonlinear Integer Arithmetic) and in the theory of bitvectors (**BV**) is included.

Keywords: Satisfiability modulo a theory, Satisfiability modulo assignment, Quantified satisfiability, Automated reasoning in arithmetic

1 Introduction

Applications of automated reasoning generate formulas involving both quantifiers and symbols defined in background theories. For example, software verification needs reasoners that decide the satisfiability of quantified formulas modulo theories such as data structures and arithmetic (e.g., [33]). Therefore, enriching first-order automated theorem provers (ATP) with built-in theories (e.g., [1, 2, 28]), integrating provers and solvers [10], and endowing SMT solvers with quantifier reasoning (e.g., [3, 14, 17–20, 22, 35, 36, 38–41]) have been and are major research objectives.

Let $\mathcal{T}$ be a single background theory. The $\mathcal{T}$-satisfiability of quantified formulas can be reduced to the $\mathcal{T}$-satisfiability of quantifier-free formulas, if $\mathcal{T}$ admits quantifier elimination (QE): for every formula φ there exists a quantifier-free formula F that is $\mathcal{T}$-equivalent to φ. Since computing F can be prohibitively expensive (e.g., exponential in linear rational arithmetic (LRA) and doubly exponential in linear integer arithmetic (LIA) [11]), QE is not a practical solution. QE is more than what is needed, because if φ is $\mathcal{T}$-satisfiable, F characterizes all solutions when one suffices for $\mathcal{T}$-satisfiability [18].

In this article we propose a practical solution in the form of a new algorithm called QSMA (*Quantified Satisfiability Modulo Assignment*). In QSMA the computation of quantifier-free *model-based under-approximations* (MBU) and *model-based over-approximations* (MBO) of quantified formulas embodies a *lazy* approach to QE, which is tailored for $\mathcal{T}$-satisfiability. MBU generates a quantifier-free formula that is true in the given model and implies in the theory the given formula. MBO generates a quantifier-free formula that is false in the given model and is implied in the theory by the given formula.

The MBU concept was preceded in the literature by that of *model-based projection* [3, 25, 26], revised to require that a model-based projection is an under-approximation [3]. Model-based projections were defined for LRA in [25], LRA, LIA, and possibly empty lists in [3], and for propositional logic, LRA, and LIA in [26]. Other instances of MBU are *model-guided generalization* for LRA [18] and *model generalization* for nonlinear real arithmetic (NRA) [24]. *Model interpolation* for NRA [24] is an instance of MBO. The more basic and older notion of *model-based quantifier instantiation* was investigated in SMT (e.g., [15, 19]), the integration of ATP and SMT (e.g., [10]), and ATP alone (e.g., [5]).

The QSMA algorithm assumes that the theory $\mathcal{T}$ is *complete*. By its recursive nature, QSMA solves a generalized form of the satisfiability problem, called *quantified SMA* (*satisfiability modulo theory and assignment*): given a formula φ with *arbitrary quantification*, and an *initial assignment* to Boolean or first-order subterms of φ, find a theory model of φ that extends the initial assignment, or report that none exists. We begin with a high-level view of the QSMA algorithm.

1.1 High-Level View of the **QSMA** Algorithm

The QSMA algorithm works by progressively assigning values to quantified variables. Consider a formula φ of the form

$$\exists\bar{x}_1.\forall\bar{x}_2.\exists\bar{x}_3\ldots F[\bar{x}_1,\bar{x}_2,\bar{x}_3,\ldots]$$

where F is quantifier-free and the variables in the tuples in brackets are those that occur in F. For example, suppose the theory is LRA, $\varphi = \exists x.\forall y.\exists z.F$ and $F = z \geq 0 \wedge x \geq 0 \wedge y + z \geq 0$. Say that QSMA assigns $x{\leftarrow}0$. Whatever value is chosen for y, the algorithm can show that φ is true in LRA by assigning $z{\leftarrow}max(0, -y)$. If $F = z \geq 0 \wedge x \geq 0 \wedge y + z \leq 0$, no matter which (non-negative) value QSMA chooses for x, it can show that φ is false in LRA by picking $y{\leftarrow}1$, because there is no value for z that satisfies $z \geq 0 \wedge z \leq -1$. For a formula that is not in prenex normal form, consider

$$\exists x.((\forall y.F[x, y]) \Rightarrow (\forall z.G[x, z]))$$

where F and G are quantifier-free, and the variables in brackets are those occurring in F and G, respectively. The QSMA algorithm sees the formula as $\exists x.((\exists y.\neg F[x, y]) \vee (\neg\exists z.\neg G[x, z]))$, and then as $\exists x.(p \vee \neg q)$, where p and q are proxy Boolean variables for the quantified subformulas. QSMA assigns values to x, p, and q. If p is assigned true, QSMA tries to extend the assignment with a value for y that satisfies $\neg F[x, y]$. If q is assigned false, QSMA tries to show that there is no value for z that satisfies $\neg G[x, z]$.

Without loss of generality ($\forall x.F[x]$ is logically equivalent to $\neg\exists x.\neg F[x]$), we consider a formula φ of the form

$$\exists \bar{x}.F[\bar{z}, \bar{x}, \bar{p}]\{p_i{\leftarrow}\exists \bar{y}_i.G_i[\bar{z}, \bar{x}, \bar{y}_i]\}_{i=1}^{k},$$

where $F[\bar{z}, \bar{x}, \bar{p}]$ is quantifier-free, because proxy Boolean variables $\bar{p} = p_1, \dots p_k$ replace the quantified subformulas $\varphi_i = \exists \bar{y}_i.G_i[\bar{z}, \bar{x}, \bar{y}_i]$. The variables occurring in the formula are listed in brackets: tuple $\bar{z}$ contains the free variables, tuple $\bar{x}$ contain the existentially quantified variables, and tuple $\bar{p}$ contains the proxy Boolean variables. The formula φ is represented by a structure called QSMA-*tree*.

Given a QSMA-tree for φ and an initial assignment to $\bar{z}$, the QSMA algorithm seeks to extend the assignment to $\bar{x}$ and $\bar{p}$ to satisfy $F[\bar{z}, \bar{x}, \bar{p}]$. If no such extension exists, formula φ is false under the initial assignment. Otherwise, there are two cases. If $k = 0$, formula φ is true under the initial assignment. If $k > 0$, the algorithm descends recursively to consider the QSMA-subtrees for $\varphi_1, \dots, \varphi_k$. If the found extension assigns true (false) to p_i, the QSMA algorithm tries to show that φ_i is true (false). If QSMA succeeds for all QSMA-subtrees, formula φ is true under the initial assignment, and the computed assignment satisfies $F[\bar{z}, \bar{x}, \bar{p}] \wedge \bigwedge_{i=1}^{k}(p_i \Leftrightarrow \varphi_i)$. Otherwise, formula φ is false under the initial assignment.

1.2 Structure and Contents

Section 2 provides basic definitions and notations used throught the article. Section 3 presents the framework of concepts of the QSMA algorithm. We introduce QSMA-*trees* to represent formulas, we define what it means to satisfy a QSMA-tree, and we prove the equivalence of satisfying a formula and its QSMA-tree. Then, we describe the key functionalities employed by the QSMA algorithm: *model extension*, by satisfiability checking modulo theory and partial assignment, *model-based under-approximation* (MBU), and *model-based over-approximation* (MBO). If a solver for the quantifier-free fragment of a complete theory offers these functionalities, the QSMA algorithm can be built on top of it, extending it to the full theory.

Section 4 presents the QSMA algorithm to solve the quantified SMA problem. We illustrate the QSMA algorithm walking the reader through its pseudocode. Then, we prove partial correctness and termination of QSMA, establishing its total correctness. Termination rests on the assumption that the MBU and MBO functions used by QSMA have *finite basis*, which means producing finitely many formulas for a given formula.

The QSMA algorithm is recursive, and it maintains throughout its execution under-approximations and over-approximations of all formulas associated to the nodes of the QSMA-tree. We present two optimizations. The first one reduces the number of recursive calls by performing a *look-ahead*, so that a path in the QSMA-tree can be handled in one shot rather than by as many recursive calls as there are nodes in the path. The second one reduces the number of stored formulas, by keeping only under-approximations. Over-approximations are computed and used in a temporary formula, but they are not kept throughout the execution. The resulting optimized variant of the QSMA algorithm, called OptiQSMA, is the subject of Section 5. We extend the QSMA framework with the notions of *look-ahead formula* and *satisfaction with look-ahead*, and we prove the equivalence of satisfying a QSMA-tree and satisfying a QSMA-tree with look-ahead. Then, we elucidate the OptiQSMA algorithm walking the reader through its pseudocode. We prove partial correctness and termination of OptiQSMA, thus showing that total correctness is preserved.

Section 6 provides concrete instances of MBU and MBO functions for LRA. The OptiQSMA algorithm is implemented in the YicesQS solver built on top of Yices 2. Section 7 contains a report on the experimental results with YicesQS in comparison with other solvers in the 2022 SMT competition, known as SMT-COMP, with updates from the 2023 and 2024 editions. Section 8 contains a discussion, including comparison with related work and directions for future work. This article was preceded by a short version [9], advertised in a one-page abstract [4].

2 Preliminaries

A signature Σ is given by a set S of sorts and a set of sorted symbols. Let $\mathcal{V} = (\mathcal{V}^s)_{s\in S}$ be a class of disjoint sets of sorted variables. Then $\Sigma[\mathcal{V}]$-formulas, Σ-sentences, and $\Sigma[\mathcal{V}]$-interpretations are defined as usual, where Σ-sentences do not depend on variables because in sentences all variables are quantified. A Σ-structure is a $\Sigma[\emptyset]$-interpretation.

We use x, y, z, w, and v for first-order variables, p for Boolean ones, $\bar{x}$, $\bar{y}$, $\bar{z}$, and $\bar{p}$ for tuples of such variables, and t, s, and u for terms. We also use $\simeq$ for equality, $\Rightarrow$ for implication, $\Leftrightarrow$ for logical equivalence, φ and ψ for formulas, F, G, and H for quantifier-free formulas, and $\bot$ and $\top$ for the false (contradiction) and true formulas, respectively. The variables that may occur in a quantifier-free formula F are listed in brackets (e.g., $F[\bar{x}]$, $F[\bar{z},\bar{x}]$, $F[\bar{z},\bar{x},\bar{p}]$). The symbol $=$ is identity, $\uplus$ is disjoint union, and $\setminus$ is set difference. On the semantic side, we use $\mathcal{M}$ for interpretations, $\mathcal{M}(x)$ for the value that $\mathcal{M}$ assigns to a first-order variable x, $\mathcal{M}(p)$ for the value that $\mathcal{M}$ assigns to a Boolean variable p, $\mathcal{M}(t)$ for the interpretation in $\mathcal{M}$ of a term t, and $\models$ for satisfaction of formulas and entailment between formulas.

$FV(\varphi)$ is the set of the variables occurring free in formula φ. Slightly abusing the notation, $FV(\varphi)$ is treated as a tuple whenever convenient. Symmetrically, a tuple of variables can be treated as a set whenever convenient, allowing us to use $\emptyset$ for the empty tuple. Also, the notation is extended to terms as in $FV(t)$ for t a term. We write $\mathcal{V}_1 \subseteq \mathcal{V}_2$ if $\mathcal{V}_1^s \subseteq \mathcal{V}_2^s$ for all $s \in S$. Given $\mathcal{V}_1$, a $\Sigma[\mathcal{V}_1]$-interpretation $\mathcal{M}_1$, and $\mathcal{V}_2$, such that $\mathcal{V}_1 \subseteq \mathcal{V}_2$, a $\Sigma[\mathcal{V}_2]$-interpretation $\mathcal{M}_2$ is an *extension* of $\mathcal{M}_1$ to $\mathcal{V}_2$, if $\mathcal{M}_2$ interprets the variables in $\mathcal{V}_2^s \setminus \mathcal{V}_1^s$ for all $s \in S$ and is otherwise identical to $\mathcal{M}_1$.

A theory $\mathcal{T}$ can be defined syntactically or semantically. In the *syntactic* definition of a theory, $\mathcal{T}$ is defined by a signature Σ and a set of Σ-sentences called $\mathcal{T}$*-axioms*. A model of $\mathcal{T}$, or $\mathcal{T}$*-model*, is a Σ-structure that satisfies the $\mathcal{T}$-axioms. A theory $\mathcal{T}$ is *consistent*, if it is impossible to derive a contradiction from the $\mathcal{T}$-axioms. A theory $\mathcal{T}$ is *complete*, if it is consistent, and for all Σ-sentences F, either F or $\neg F$ is provable from the $\mathcal{T}$-axioms. In the *semantic* definition of a theory, $\mathcal{T}$ is defined by a signature Σ and a set of Σ-structures called $\mathcal{T}$*-models*. A theory $\mathcal{T}$ is *consistent*, if the set of the $\mathcal{T}$-models is not empty. A theory $\mathcal{T}$ is *complete*, if it is consistent, and for all Σ-sentences F, either F or $\neg F$ is true in all the $\mathcal{T}$-models. In either definition, a $\mathcal{T}[\mathcal{V}]$-model is a $\Sigma[\mathcal{V}]$-interpretation that is a $\mathcal{T}$-model when the interpretation of variables is ignored.

In this article we adopt the *semantic* definition of a theory, and we deal with a *single* theory $\mathcal{T}$ that has a *unique* $\mathcal{T}$-model $\mathcal{M}_0$, so that the interpretation of every symbol except variables is fixed. Therefore $\mathcal{T}$ is complete, for Σ-sentences $\mathcal{T}$-validity, $\mathcal{T}$-satisfiability, and truth in $\mathcal{M}_0$ coincide, all $\mathcal{T}[\mathcal{V}]$-models are extensions of $\mathcal{M}_0$, and a $\mathcal{T}$-satisfiability procedure is concerned only with assignments to variables.

A *conservative theory extension* $\mathcal{T}^+$ of $\mathcal{T}$ adds to Σ special constants, called *values*, to name elements in the domain of $\mathcal{M}_0$ as needed. Conservative means that a $\mathcal{T}$-satisfiable formula is also $\mathcal{T}^+$-satisfiable. Since there are one theory $\mathcal{T}$ and one signature Σ, we write formula for $\Sigma[\mathcal{V}]$-formula, model for $\mathcal{T}$-model or $\mathcal{T}[\mathcal{V}]$-model, satisfaction for $\mathcal{T}$-satisfaction, and satisfiability for $\mathcal{T}$-satisfiability.

The *quantified SMA problem* for theory $\mathcal{T}$ asks whether $\mathcal{M}_0 \models \varphi$ for an arbitrary formula φ and an initial assignment of values to the variables in $FV(\varphi)$. Formulas have the form $\varphi = \exists \bar{x}. F[\bar{z}, \bar{x}, \bar{p}]\{p_i \leftarrow \exists \bar{y}_i. G_i[\bar{z}, \bar{x}, \bar{y}_i]\}_{i=1}^k$ described in the introduction, where $FV(\varphi) = \bar{z}$. If $FV(\varphi) = \emptyset$, we still get SMA problems when considering subformulas under an assignment to initially existentially quantified variables.

3 The QSMA Framework

The QSMA algorithm works with a tree representation of a formula φ. A node n in the tree is labeled with a pair $(\bar{x}, F)$, where $\bar{x}$ is a tuple of first-order variables, called the *local variables* of node n, and F is a quantifier-free formula. The local variables are implicitly existentially quantified: they are existentially quantified variables whose quantifers have been removed, so that they are locally free, so to speak, and can be assigned by the algorithm. An arc from a node n to a child node b is labeled with a Boolean variable p. This Boolean variable stands as a *proxy* for the quantified subformula represented by the subtree rooted at node b. Therefore, the Boolean variable p is also considered a proxy of node b itself.

A formula φ may have free variables $FV(\varphi) = \bar{z}$, whose assignment is given initially as part of the SMA problem instance. These variables are called *rigid*, because their assignments do not change during the tree traversal. As the algorithm traverses the tree, the local variables of a node n are *rigid* from the viewpoint of a node b that is a child (or a descendant) of n: their assignments do not change during the traversal of the subtree rooted at b. Therefore, we represent a formula φ as a pair formed by a tuple of rigid variables and a labeled tree. Slightly abusing the terminology, we call this pair a QSMA-*tree*. The root of a tree T is denoted *root*(T).

Definition 1 (QSMA-tree)**.** *Given formula $\varphi = \exists\bar{x}.F[\bar{z},\bar{x},\bar{p}]\{p_i\leftarrow\exists\bar{y}_i.G_i[\bar{z},\bar{x},\bar{y}_i]\}_{i=1}^{k}$, with free variables $FV(\varphi) = \bar{z}$ and distinct quantified subformulas $\varphi_i = \exists\bar{y}_i.G_i[\bar{z},\bar{x},\bar{y}_i]$, $1 \leq i \leq k$, the* QSMA-tree *for φ is the pair $\mathcal{G} = (\bar{z}, T)$, where $\bar{z}$ is called the tuple of the rigid variables of $\mathcal{G}$, and T is a labeled tree defined inductively as follows:*

- *If $k = 0$, the tree T consists of a single root node r labeled $(\bar{x}, F[\bar{z},\bar{x}])$ where $\bar{x}$ are the local variables of r;*
- *If $k > 0$, for all i, $1 \leq i \leq k$, let $\mathcal{G}_i = ((\bar{z},\bar{x}), T_i)$ be the* QSMA-*tree for φ_i, where root(T_i) is a node b_i labeled $(\bar{y}_i, G_i[\bar{z},\bar{x},\bar{y}_i])$. Then T is the tree formed by a new root node r, labeled $(\bar{x}, F[\bar{z},\bar{x},\bar{p}])$, with k outgoing arcs, labeled $p_1,\ldots,p_k$, connecting node r to children nodes $b_1,\ldots,b_k$, respectively.*

The attribute "distinct" in the above definition implies that if a subformula occurs more than once in the formula, the same proxy variable is used for all occurrences. In the following we may use also c, d, and e for children nodes, and q, o, and l for Boolean proxy variables.

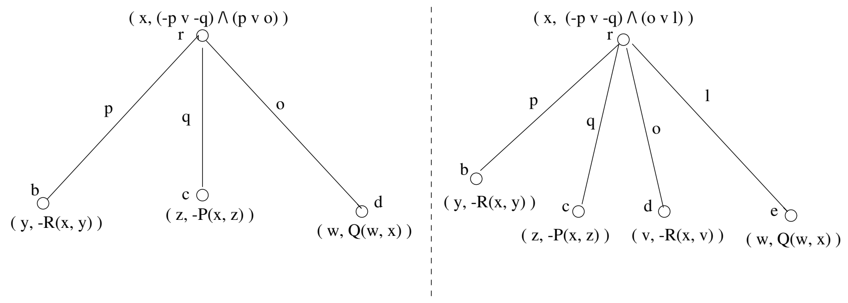

Fig. 1 The QSMA-trees for formulas φ_1 (on the left) and φ_2 (on the right) in Example 1

Example 1. *Consider a formula where a quantified subformula appears twice: $\exists x.\{[(\forall y.R(x,y)) \vee (\forall z.P(x,z))] \wedge [(\forall y.R(x,y)) \Rightarrow (\exists w.Q(w,x))]\}$. Conversion of universal quantifiers into existential ones and replacement of implication with disjunction produce $\varphi_1 = \exists x.\{[(\neg(\exists y.\neg R(x,y))) \vee (\neg(\exists z.\neg P(x,z)))] \wedge [(\exists y.\neg R(x,y)) \vee (\exists w.Q(w,x))]\}$. The* QSMA-*tree for φ_1 is shown in Figure 1 on the left. The structure for φ_1 is a tree, not a DAG (directed acyclic graph). It would be a DAG, if it had two outgoing arcs, both labeled p, from root r to child node b. However, this duplication is unnecessary, because the information that p occurs twice is recorded in* *the formula in the label of root r. If variables were standardized apart, one would get $\varphi_2 = \exists x.\{[(\neg(\exists y.\neg R(x,y))) \vee (\neg(\exists z.\neg P(x,z)))] \wedge [(\exists v.\neg R(x,v)) \vee (\exists w.Q(w,x))]\}$. The* QSMA*-tree for φ_2 is shown in Figure 1 on the right. If quantified variables are standardized apart, every quantified subformulas is guaranteed to appear only once. However, p and o represent the same subproblem: given an assignment to x find an assignment to y (v) such that $\neg R(x,y)$ ($\neg R(x,v)$). If a subformula occurs more than once, standardization of variables is to be avoided, in order to avoid generating redundant subproblems.*

The *ancestors* of a node n in tree T are the nodes on the unique path from $root(T)$ to n excluding n itself. If node n in T is labeled $(\bar{x}, F)$, its k outgoing arcs are labeled $p_1, \ldots, p_k$, and $\bar{x}_1, \ldots, \bar{x}_m$ are the local variables of the ancestors of n, then $FV(F) \subseteq \{\bar{z}, \bar{x}_1, \ldots, \bar{x}_m, \bar{x}, p_1, \ldots, p_k\}$. The set of the *assignable variables at node* n is $Var(n) = \bar{x} \uplus \{p_1, \ldots, p_k\}$. In words, the assignable variables at node n are the local variables of n and the proxy Boolean variables labeling the outgoing arcs connecting n to its children nodes. The set of the *rigid variables at node* n is $Rigid(n) = \bar{z} \uplus \bar{x}_1 \uplus \ldots \uplus \bar{x}_m$. In words, the rigid variables at node n are the rigid variables of the whole QSMA-tree $\mathcal{G} = (\bar{z}, T)$ plus the local variables of n's ancestors. Thus, $FV(F) \subseteq Rigid(n) \cup Var(n)$. Since the root has no ancestors, the set of the rigid variables at the root contains only $\mathcal{G}$'s rigid variables: $Rigid(root(T)) = \bar{z}$. In terms of notation, we use $\mathcal{G}_n$ for the QSMA-subtree rooted at node n and T_n for its second component, so that $\mathcal{G}_n = (Rigid(n), T_n)$. For a node n with label $(\bar{x}, F)$, the components of the label are denoted $n.\bar{x}$ and $n.F$. The label of the arc from node n to a child node b is denoted $b.p$.

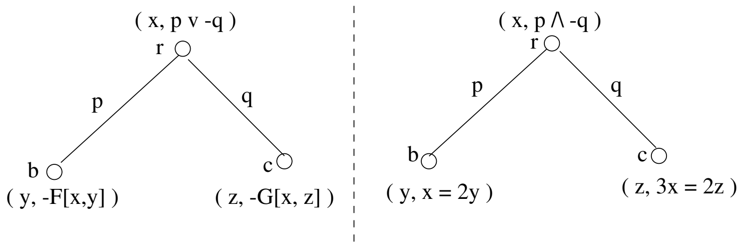

Fig. 2 The QSMA-trees for Example 2 on the left and for Example 3 on the right

Example 2. *Reconsider formula $\exists x.((\forall y.F[x,y]) \Rightarrow (\forall z.G[x,z]))$ from Section 1.1. Elimination of implication and conversion of the second universal quantifier to existential by double negation introduction yields $\varphi = \exists x.((\exists y.\neg F[x,y]) \vee (\neg\exists z.\neg G[x,z]))$. The* QSMA*-tree $\mathcal{G}$ for φ appears in Figure 2 on the left. For the root r we have $FV(r.F) \subseteq \{x,p,q\}$, $Var(r) = \{x,p,q\}$, and $Rigid(r) = \emptyset$. For children nodes b and c we have $FV(b.F) \subseteq \{x,y\}$, $FV(c.F) \subseteq \{x,z\}$, $Var(b) = \{y\}$, $Var(c) = \{z\}$, and $Rigid(b) = Rigid(c) = \{x\}$.*

Example 3. *Consider formula $\forall x.((\exists y.(x \simeq 2{\cdot}y)) \Rightarrow (\exists z.(3{\cdot}x \simeq 2{\cdot}z)))$ in* LRA*. A double negation converts the universal quantifier yielding $\neg(\exists x.((\exists y.(x \simeq 2{\cdot}y)) \wedge (\forall z.(3{\cdot}x \not\simeq 2{\cdot}z))))$. Again, a double negation converts the universal quantifier producing $\neg(\exists x.((\exists y.(x \simeq 2{\cdot}y)) \wedge (\neg(\exists z.(3{\cdot}x \simeq 2{\cdot}z)))))$. Let $\varphi = \exists x.((\exists y.(x \simeq 2{\cdot}y)) \wedge (\neg(\exists z.(3{\cdot}x \simeq$*

$2{\cdot}z))))$. *The original formula is true in* LRA *iff* φ *is false in* LRA*. The* QSMA*-tree* $\mathcal{G}$ *for* φ *appears in Figure 2 on the right. The variable sets are the same as in Example 2.*

Conversely, given a QSMA-tree $\mathcal{G} = (\bar{z}, T)$, we can associate a formula $n.\psi$ to every node n in T and hence to the QSMA-subtree $\mathcal{G}_n = (\mathit{Rigid}(n), T_n)$.

Definition 2 (Formula at a node)**.** *Given a* QSMA*-tree* $\mathcal{G} = (\bar{z}, T)$*, for all nodes* n *of* T*, the* formula $n.\psi$ at node n *is defined inductively as follows:*

- *If* n *is a leaf labeled* $(\bar{x}, F[\bar{z}, \bar{x}])$*, then* $n.\psi = \exists\bar{x}.F[\bar{z}, \bar{x}]$*;*
- *If* n *has label* $(\bar{x}, F[\bar{z}, \bar{x}, \bar{p}])$ *and outgoing arcs labeled* $p_1, \ldots, p_k$ $(k > 0)$ *connecting* n *to children nodes* $b_1, \ldots, b_k$*, let* $b_1.\psi, \ldots, b_k.\psi$ *be the formulas at nodes* $b_1, \ldots, b_k$*, respectively. Then* $n.\psi = \exists\bar{x}.F[\bar{z}, \bar{x}, \bar{p}]\{p_i \leftarrow b_i.\psi\}_{i=1}^{k}$*.*

If $\mathcal{G} = (\bar{z}, T)$ is the QSMA-tree for φ and $r = \mathit{root}(T)$, then $r.\psi = \varphi$.

Example 4. *For the* QSMA*-tree in Example 3 (Figure 2, right),* $b.\psi = \exists y.(x \simeq 2{\cdot}y)$*,* $c.\psi = \exists z.(3{\cdot}x \simeq 2{\cdot}z)$*, and* $r.\psi = \exists x.((\exists y.(x \simeq 2{\cdot}y)) \wedge (\neg(\exists z.(3{\cdot}x \simeq 2{\cdot}z)))) = \varphi$*.*

Since the input formula φ is represented as a QSMA-tree $\mathcal{G} = (\bar{z}, T)$, the problem of satisfying φ becomes the problem of satisfying $\mathcal{G}$. Therefore, we define *satisfaction of a* QSMA*-tree* next. Slightly abusing the notation, we use $\models$ also for satisfaction of QSMA-trees. Recall that $\mathcal{M}_0$ is the unique model of the assumed theory $\mathcal{T}$.

Definition 3 (Satisfaction of a QSMA-tree)**.** *Given a* QSMA*-tree* $\mathcal{G} = (\bar{z}, T)$ *with root node* $r = \mathit{root}(T)$*, and an extension* $\mathcal{M}$ *of* $\mathcal{M}_0$ *to the set* $\mathit{Rigid}(r) = \bar{z}$ *of the rigid variables at node* r*,* $\mathcal{M} \models \mathcal{G}$ *if there exists an extension* $\mathcal{M}'$ *of* $\mathcal{M}$ *to the set* $\mathit{Var}(r)$ *of the assignable variables at node* r*, such that (i)* $\mathcal{M}' \models r.F$*, and (ii) for all children nodes* b *of* r*,* $\mathcal{M}'(b.p) = \mathsf{true}$ *iff* $\mathcal{M}' \models \mathcal{G}_b$*.*

The QSMA algorithm works by traversing the QSMA-tree $\mathcal{G} = (\bar{z}, T)$, and at each node n in T it assigns the assignable variables in $\mathit{Var}(n) = \bar{x} \uplus \{p_1, \ldots, p_k\}$. This assignment corresponds to the extension $\mathcal{M}'$ in Definition 3. Note how this assignment (and $\mathcal{M}'$ in Definition 3) contains *both first-order (i.e., non-Boolean) and Boolean assignments*. Let node b be a child of n: the Boolean variable $b.p$ labeling the arc from n to b is a proxy for the quantified subformula $b.\psi$ of the formula $n.\psi$. If $\mathcal{M}'(b.p) = \mathsf{true}$, the aim of the algorithm is to show that $b.\psi$ is true, and if $\mathcal{M}'(b.p) = \mathsf{false}$, the aim is to show that $b.\psi$ is false. Therefore Condition (ii) in Definition 3 says $\mathcal{M}' \models \mathcal{G}_b$ if $\mathcal{M}'(b.p) = \mathsf{true}$ and $\mathcal{M}' \not\models \mathcal{G}_b$ if $\mathcal{M}'(b.p) = \mathsf{false}$. The next theorem shows that satisfying a formula φ and satisfying the QSMA-tree for φ correspond.

Theorem 1 (Equivalence of satisfaction of formulas and QSMA-trees)**.** *For all formulas* φ *with* $FV(\varphi) = \bar{z}$*, for all models* $\mathcal{M}$ *extending* $\mathcal{M}_0$ *to* $\bar{z}$*, if* $\mathcal{G}$ *is the* QSMA*-tree for* φ *then* $\mathcal{M} \models \mathcal{G}$ *iff* $\mathcal{M} \models \varphi$*.*

Proof. The formula φ and the QSMA-tree $\mathcal{G} = (\bar{z}, T)$ are as in Definition 1. The proof is by induction on the number k of the distinct quantified subformulas φ_i of φ.
Base case: $k = 0$ so that $\varphi = \exists\bar{x}.F[\bar{z}, \bar{x}]$ and T consists of the single node r with label $(\bar{x}, F[\bar{z}, \bar{x}])$. By Definition 3, $\mathcal{M} \models \mathcal{G}$ iff there exists an extension $\mathcal{M}'$ of $\mathcal{M}$ to $\mathit{Var}(r) = \bar{x}$ such that $\mathcal{M}' \models F[\bar{z}, \bar{x}]$, that is, iff $\mathcal{M} \models \varphi$.

Induction hypothesis: $k \geq 0$, and for all i, $1 \leq i \leq k$, for all models $\mathcal{M}$ extending $\mathcal{M}_0$ to $\mathit{Rigid}(b_i) = \bar{z} \uplus \bar{x}$, $\mathcal{M} \models \mathcal{G}_i$ iff $\mathcal{M} \models \varphi_i$.
Induction step: we distinguish the two directions.
$\Rightarrow$) Let $\mathcal{M}$ be an extension of $\mathcal{M}_0$ to $\mathit{Rigid}(r) = \bar{z}$, such that $\mathcal{M} \models \mathcal{G}$. By Definition 3, there exists an extension $\mathcal{M}'$ of $\mathcal{M}$ to $\mathit{Var}(r)$ such that $\mathcal{M}' \models F[\bar{z}, \bar{x}, \bar{p}]$ and for all i, $1 \leq i \leq k$, $\mathcal{M}'(p_i) = \mathsf{true}$ iff $\mathcal{M}' \models \mathcal{G}_i$. By induction hypothesis, $\mathcal{M}'(p_i) = \mathsf{true}$ iff $\mathcal{M}' \models \varphi_i$. Therefore, $\mathcal{M}' \models F[\bar{z}, \bar{x}, \bar{p}] \wedge \bigwedge_{i=1}^{k} p_i \Leftrightarrow \varphi_i$, and hence $\mathcal{M} \models \varphi$.
$\Leftarrow$) Let $\mathcal{M}$ be an extension of $\mathcal{M}_0$ to $\mathit{Rigid}(r) = \bar{z}$, such that $\mathcal{M} \models \varphi$. Under $\mathcal{M}$'s interpretation of $\bar{z} \uplus \bar{x}$, φ is equisatisfiable to $\psi = F[\bar{z}, \bar{x}, \bar{p}] \wedge \bigwedge_{i=1}^{n} p_i \Leftrightarrow \varphi_i$. Let $\mathcal{M}'$ be a model of ψ: $\mathcal{M}'$ is a model of $F[\bar{z}, \bar{x}, \bar{p}]$ such that $\mathcal{M}'(p_i) = \mathsf{true}$ iff $\mathcal{M}' \models \varphi_i$. By induction hypothesis, $\mathcal{M}'(p_i) = \mathsf{true}$ iff $\mathcal{M}' \models \mathcal{G}_i$. By Definition 3, $\mathcal{M} \models \mathcal{G}$. $\square$

Checking whether $\mathcal{M} \models \mathcal{G}$ by testing all possible extensions $\mathcal{M}'$ would not do, because for most theories (e.g., LRA) there is an infinite number of extensions. We need a way to weed out large parts of the space of candidate models. Let $[\![\varphi]\!]$ denote the set of models of formula φ. We introduce *under-approximations* and *over-approximations* of φ in order to under-approximate and over-approximate $[\![\varphi]\!]$.

Definition 4 (Under-approximation)**.** *Let φ be a formula with $FV(\varphi) = \bar{z}$. A quantifier-free formula U with $FV(U) = \bar{z}$ is an* under-approximation *of φ, if for all extensions $\mathcal{M}$ of $\mathcal{M}_0$ to $\bar{z}$, $\mathcal{M} \models U$ implies $\mathcal{M} \models \varphi$.*

Note that $FV(U) = FV(\varphi)$ holds, even if the process of computing U from φ eliminates variables, because the variables that get eliminated are existentially quantified (universal quantifiers are converted into existential ones during pre-processing). This remark applies also to the next definition.

Definition 5 (Over-approximation)**.** *Let φ be a formula with $FV(\varphi) = \bar{z}$. A quantifier-free formula O with $FV(O) = \bar{z}$ is an* over-approximation *of φ, if for all extensions $\mathcal{M}$ of $\mathcal{M}_0$ to $\bar{z}$, $\mathcal{M} \models \varphi$ implies $\mathcal{M} \models O$.*

The key point is that $\mathcal{M} \in [\![U]\!]$ implies $\mathcal{M} \in [\![\varphi]\!]$ and $\mathcal{M} \in [\![\varphi]\!]$ implies $\mathcal{M} \in [\![O]\!]$, so that $[\![U]\!] \subseteq [\![\varphi]\!] \subseteq [\![O]\!]$. Let $\mathcal{G} = (\bar{z}, T)$ be the QSMA-tree for φ, and U and O under- and over-approximations of φ, respectively. Then, $\mathcal{M} \models U$ implies $\mathcal{M} \models \varphi$ which implies $\mathcal{M} \models \mathcal{G}$ (by Theorem 1). Thus, satisfying an under-approximation is a *sufficient condition* to have a solution. On the other hand, $\mathcal{M} \models \mathcal{G}$ implies $\mathcal{M} \models \varphi$ (by Theorem 1) which implies $\mathcal{M} \models O$. Thus, satisfying an over-approximation is a *necessary condition* to have a solution.

In order to construct approximations of formulas, we assume to have a solver for theory $\mathcal{T}$ (and model $\mathcal{M}_0$) offering the following functions:

- *Model extension*: A function SMA such that for all formulas $\exists\bar{x}.F[\bar{z}, \bar{x}]$, where $F[\bar{z}, \bar{x}]$ is quantifier-free, and all extensions $\mathcal{M}$ of $\mathcal{M}_0$ to $\bar{z}$, the call $\mathsf{SMA}(F[\bar{z}, \bar{x}], \mathcal{M})$ returns either an extension $\mathcal{M}'$ of $\mathcal{M}$ to $\bar{x}$ such that $\mathcal{M}' \models F[\bar{z}, \bar{x}]$, or *nil* if no such extension exists.
- *Model-based under-approximation*: A function MBU such that for all formulas $\exists\bar{x}.F[\bar{z}, \bar{x}]$, where $F[\bar{z}, \bar{x}]$ is quantifier-free, and all extensions $\mathcal{M}$ of $\mathcal{M}_0$ to $\bar{z}$ such

that $\mathcal{M} \models \exists \bar{x}.F[\bar{z}, \bar{x}]$, the call $\mathsf{MBU}(F[\bar{z}, \bar{x}], \bar{x}, \mathcal{M})$ returns a quantifier-free formula $U[\bar{z}]$ such that $\mathcal{M} \models U[\bar{z}]$ and $\mathcal{T} \models U[\bar{z}] \Rightarrow (\exists \bar{x}.F[\bar{z}, \bar{x}])$.
- *Model-based over-approximation*: A function MBO such that for all formulas $\exists \bar{x}.F[\bar{z}, \bar{x}]$, where $F[\bar{z}, \bar{x}]$ is quantifier-free, and all extensions $\mathcal{M}$ of $\mathcal{M}_0$ to $\bar{z}$ such that $\mathcal{M} \not\models \exists \bar{x}.F[\bar{z}, \bar{x}]$, the call $\mathsf{MBO}(F[\bar{z}, \bar{x}], \bar{x}, \mathcal{M})$ returns a quantifier-free formula $O[\bar{z}]$ such that $\mathcal{M} \not\models O[\bar{z}]$ and $\mathcal{T} \models (\exists \bar{x}.F[\bar{z}, \bar{x}]) \Rightarrow O[\bar{z}]$.

The function SMA performs a *satisfiability check modulo theory and assignment*: it takes the quantifier-free formula $F[\bar{z}, \bar{x}]$, obtained by dropping the existential quantifiers in $\exists \bar{x}.F[\bar{z}, \bar{x}]$, and an assignment $\mathcal{M}$ assigning values to the variables in $\bar{z}$; and it returns either an assignment $\mathcal{M}'$ that satisfies $F[\bar{z}, \bar{x}]$ by extending $\mathcal{M}$ with values for the variables in $\bar{x}$, or *nil* if no such extension exists.

The MBU and MBO functions take as arguments the formula $F[\bar{z}, \bar{x}]$, where $\bar{z}$ are the free variables in the original formula $\varphi = \exists \bar{x}.F[\bar{z}, \bar{x}]$, the tuple $\bar{x}$ of the existentially quantified variables in φ, and the extension $\mathcal{M}$ that assigns values to the variables in $\bar{z}$, and produce formulas $U[\bar{z}]$ and $O[\bar{z}]$, respectively, where the variables $\bar{z}$ still occur free, whereas the variables $\bar{x}$ are gone. This matches the requirements $FV(U) = FV(\varphi)$ and $FV(O) = FV(\varphi)$ of Definitions 4 and 5, respectively. The key properties $\mathcal{T} \models U[\bar{z}] \Rightarrow (\exists \bar{x}.F[\bar{z}, \bar{x}])$ and $\mathcal{T} \models (\exists \bar{x}.F[\bar{z}, \bar{x}]) \Rightarrow O[\bar{z}]$ in the specifications of MBU and MBO ensure that the output formulas $U[\bar{z}]$ and $O[\bar{z}]$ be an under-approximation and an over-approximation, respectively, of $\exists \bar{x}.F[\bar{z}, \bar{x}]$.

Since $U[\bar{z}]$ is true in model $\mathcal{M}$ and it implies $\exists \bar{x}.F[\bar{z}, \bar{x}]$ in the theory, formula $U[\bar{z}]$ can be seen as a *theory interpolant between model and formula*. The name *model-based projection* [3, 25, 26] descends from viewing the generation of the quantifier-free formula $U[\bar{z}]$ as a projection over the existentially quantified variables $\bar{x}$. The names *model-guided generalization* [18] or *model generalization* [24, 27] stem from the consideration that $U[\bar{z}]$ may have other models in addition to $\mathcal{M}$. Since $O[\bar{z}]$ follows from $\exists \bar{x}.F[\bar{z}, \bar{x}]$ and it is false in $\mathcal{M}$, formula $O[\bar{z}]$ can be seen as a *reverse interpolant between formula and model*, which explains the name *model interpolant* [24].

4 The QSMA Algorithm and Its Correctness

Let $\mathcal{G} = (\bar{z}, T)$ be the QSMA-tree for input formula φ with $FV(\varphi) = \bar{z}$. Given a model $\mathcal{M}$ extending $\mathcal{M}_0$ to the variables in $\bar{z}$, the QSMA algorithm determines whether $\mathcal{M} \models \mathcal{G}$. Suppose that U and O are under- and over-approximations of φ, respectively. Figure 3 shows $[\![U]\!]$, $[\![\varphi]\!]$, and $[\![O]\!]$ as bubbles. The idea of the QSMA algorithm is to zoom in on a model of φ, by progressively weakening U, so that the $[\![U]\!]$ bubble inflates (outward arrows in Figure 3), and progressively strengthening O, so that the $[\![O]\!]$ bubble deflates (inward arrows in Figure 3).

4.1 Description of the QSMA Algorithm

The QSMA algorithm operates recursively maintaining under- and over-approximations for all subformulas of the input formula φ. For all nodes n of T in the QSMA-tree $\mathcal{G} = (\bar{z}, T)$ for φ, QSMA builds under- and over-approximations $n.U$ and $n.O$ of $n.\psi$ (the formula at node n), progressively weakening $n.U$ and strengthening

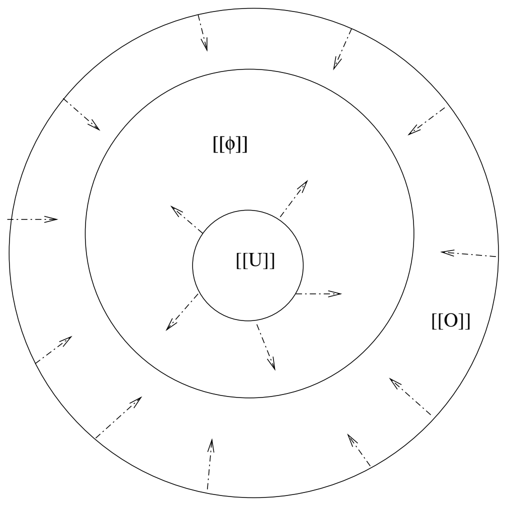

Fig. 3 The model sets $[\![U]\!] \subseteq [\![\varphi]\!] \subseteq [\![O]\!]$ and their evolution as formulas U and O evolve

$n.O$. The weakening of $n.U$ is done by introducing a disjunction with an MBU (a formula produced by a call to the MBU function). The strengthening of $n.O$ is done by introducing a conjunction with an MBO (a formula produced by a call to the MBO function). The goal is that $\mathcal{M}$ satisfies $n.U \vee \neg n.O$. As soon as $\mathcal{M}$ satisfies $n.U$, we know that $\mathcal{M} \models \mathcal{G}_n$, because satisfying $n.U$ is a sufficient condition. As soon as $\mathcal{M}$ satisfies $\neg n.O$, and hence does not satisfy $n.O$, we know that $\mathcal{M} \not\models \mathcal{G}_n$, because satisfying $n.O$ is a necessary condition.

Algorithm 1 Pseudocode of the main function of QSMA

```
@pre: $\mathcal{G} = (\bar{z}, T)$: QSMA-tree for $\varphi$ with $FV(\varphi) = \bar{z}$; $\mathcal{M}$: extension of $\mathcal{M}_0$ to $\bar{z}$
@post: $rv$ iff $\mathcal{M} \models \mathcal{G}$     ($rv$ is "returned value")
1: function QSMA($\mathcal{M}$, $T$)
2:     for all nodes $n$ in $T$ do
3:         $n.U \leftarrow \bot$
4:         $n.O \leftarrow \top$
5:     return subtreeIsSolved($root(T)$, $\mathcal{M}$)
```

The main function QSMA (Algorithm 1) initializes $n.U$ to $\bot$ and $n.O$ to $\top$ for all nodes n of T. Indeed, $\bot$ is an under-approximation of all formulas, and it is the identity for disjunction. Dually, $\top$ is an over-approximation of all formulas, and it is the identity for conjunction. Then QSMA calls[1] the function `subtreeIsSolved` (Algorithm 2) with $root(T)$ and $\mathcal{M}$ as arguments.

Function `subtreeIsSolved` takes a node n and a model $\mathcal{M}$ extending $\mathcal{M}_0$ to $Rigid(n)$ (precondition of Algorithm 2), maintains the invariant $\forall b \in T.\ [\![b.U]\!] \subseteq$

[1]Unless otherwise stated, parameters in pseudocode are passed by value.

Algorithm 2 Pseudocode of the auxiliary functions of QSMA

```
@pre: $\mathcal{M}$: extension of $\mathcal{M}_0$ to $Rigid(n)$ and $\forall b \in T.\ [\![b.U]\!] \subseteq [\![b.\psi]\!] \subseteq [\![b.O]\!]$
@post: $\forall b \in T.\ [\![b.U]\!] \subseteq [\![b.\psi]\!] \subseteq [\![b.O]\!]$ and $\mathcal{M} \models (n.U \vee \neg n.O)$ and ($rv$ iff $\mathcal{M} \models n.U$)
        ($\neg rv$ iff $\mathcal{M} \models \neg n.O$) and ($rv$ iff $\mathcal{M} \models \mathcal{G}_n$)
1:  function SUBTREEISSOLVED($n$, $\mathcal{M}$)
2:      if $\mathcal{M} \models n.U$ then
3:          return $true$
4:      if $\mathcal{M} \models \neg n.O$ then
5:          return $false$
6:      while true do
7:          $L \leftarrow n.F \wedge \bigwedge_{n \to b}((b.p \Rightarrow b.O) \wedge (\neg b.p \Rightarrow \neg b.U))$
8:          $\mathcal{M}' \leftarrow$ SMA($L$, $\mathcal{M}$)
9:          if $\mathcal{M}' = nil$ then
10:             $n.O \leftarrow n.O \wedge$ MBO($L, FV(L) \setminus Rigid(n), \mathcal{M}$)
11:             return $false$
12:         if SOLUTIONFORALLCHILDREN($n$, $\mathcal{M}'$) then
13:             $L' \leftarrow n.F \wedge \bigwedge_{n \to b}((b.p \Rightarrow b.U) \wedge (\neg b.p \Rightarrow \neg b.O))$
14:             $n.U \leftarrow n.U \vee$ MBU($L', FV(L') \setminus Rigid(n), \mathcal{M}$)
15:             return $true$
16:
17: function SOLUTIONFORALLCHILDREN($n$, $\mathcal{M}$)
18:     for all children $b$ of $n$ do
19:         if $\mathcal{M}(b.p) \neq$ SUBTREEISSOLVED($b$, $\mathcal{M}$) then
20:             return $false$
21:     return $true$
```

$[\![b.\psi]\!] \subseteq [\![b.O]\!]$ (both pre and postcondition), and determines whether $\mathcal{M} \models \mathcal{G}_n$ (postcondition). It returns *true* if $\mathcal{M} \models n.U$, and *false* if $\mathcal{M} \models \neg n.O$ (see lines 2-5 in the pseudocode and the postconditions). Otherwise (i.e., $\mathcal{M} \models \neg n.U \wedge n.O$), it enters a loop whose body contains the following steps:

1. Build a formula L as the conjunction of $n.F$ and a formula for every child node b of node n, denoted $n \to b$ (line 7 in Algorithm 2). If $b.U = \bot$ and $b.O = \top$ for all b's (e.g., right after initialization), L gets $n.F$, as the rest of the conjunction reduces to $\top$. If $b.U$ and $b.O$ are nontrivial, the shape of the formula for each b is explained by considering a model of L and hence in the next step.
2. Invoke the SMA function to search for an extension $\mathcal{M}'$ of $\mathcal{M}$ to $Var(n)$ such that $\mathcal{M}' \models L$ (line 8). For all children b of n, $b.p \in Var(n)$ and $\mathcal{M}'$ assigns a Boolean value to $b.p$. If $\mathcal{M}'(b.p) =$ true, the subformula for b in L reduces to $b.O$, so that $\mathcal{M}' \models L$ implies $\mathcal{M}' \models b.O$. Since $\mathcal{M}'(b.p) =$ true, QSMA seeks to satisfy $b.\psi$. Since $[\![b.\psi]\!] \subseteq [\![b.O]\!]$, QSMA aims at satisfying $b.\psi$, by starting at least from a model of $b.O$. If $\mathcal{M}'(b.p) =$ false, the subformula for b in L reduces to $\neg b.U$, so that $\mathcal{M}' \models L$ implies $\mathcal{M}' \models \neg b.U$. Since $\mathcal{M}'(b.p) =$ false, QSMA seeks to falsify $b.\psi$. Since $[\![b.U]\!] \subseteq [\![b.\psi]\!]$, QSMA aims at falsifying $b.\psi$ by starting at least from a model of $\neg b.U$. The proof of partial correctness of `subtreeIsSolved` will show that the

existence of an $\mathcal{M}'$ such that $\mathcal{M}' \models L$ is a necessary condition for $\mathcal{M} \models \mathcal{G}_n$ (cf. the induction step in the proof of Theorem 2).

3. If SMA returns *nil*, no extension $\mathcal{M}'$ of $\mathcal{M}$ to $\mathit{Var}(n)$ satisfies $n.F$, and hence $\mathcal{M} \not\models \mathcal{G}_n$ by Definition 3. Then `subtreeIsSolved` updates $n.O$ to its conjunction with $\mathsf{MBO}(L, FV(L) \setminus \mathit{Rigid}(n), \mathcal{M})$ (line 10). Since $\mathcal{M} \not\models L$, by MBO's specification we know that $\mathcal{M} \not\models \mathsf{MBO}(L, FV(L) \setminus \mathit{Rigid}(n), \mathcal{M})$. This update ensures that $\mathcal{M} \not\models n.O$, so that $\mathcal{M} \models \neg n.O$. Then `subtreeIsSolved` returns *false* (line 11).
4. Otherwise, we have an extension $\mathcal{M}'$ that satisfies L and hence $n.F$, so that there is potential for $\mathcal{M} \models \mathcal{G}_n$. Function `solutionForallChildren` is invoked to determine whether this is the case.
5. The function `solutionForallChildren` calls `subtreeIsSolved` for every child node b of node n. As soon as it finds a child b such that $\mathcal{M}(b.p) = \mathsf{true}$ and the call `subtreeIsSolved`(b,$\mathcal{M}$) returns *false*, or $\mathcal{M}(b.p) = \mathsf{false}$ and the call `subtreeIsSolved`(b,$\mathcal{M}$) returns *true*, it returns *false*, because it found a QSMA-subtree where the candidate model $\mathcal{M}$ fails (see Definition 3). If this does not happen, `solutionForallChildren` returns *true*.
6. If `solutionForallChildren` returns *true*, `subtreeIsSolved` builds a formula L' as the conjunction of $n.F$ and a formula for every child b of n (line 13). If $\mathcal{M}'(b.p) = \mathsf{true}$, the subformula for b in L' reduces to $b.U$. If $\mathcal{M}'(b.p) = \mathsf{false}$, the subformula for b in L' reduces to $\neg b.O$. The proof of partial correctness of `subtreeIsSolved` will show that (I) $\mathcal{M}' \models L'$ and (II) $\mathcal{M}' \models L'$ is a sufficient condition for $\mathcal{M} \models \mathcal{G}_n$ (cf. the induction step in the proof of Theorem 2). Then `subtreeIsSolved` updates $n.U$ to its disjunction with $\mathsf{MBU}(L', FV(L') \setminus \mathit{Rigid}(n), \mathcal{M})$ (line 14). Since $\mathcal{M}' \models L'$, by MBU's specification we know that $\mathcal{M}' \models \mathsf{MBU}(L', FV(L') \setminus \mathit{Rigid}(n), \mathcal{M})$. This update ensures that $\mathcal{M}' \models n.U$. Then `subtreeIsSolved` returns *true* (line 15).
7. If `solutionForallChildren` returns *false*, the control returns to line 7. Suppose that `solutionForallChildren` returned *false*, because it found a child node b of node n such that $\mathcal{M}(b.p) = \mathsf{true}$ and `subtreeIsSolved`(b,$\mathcal{M}$) returned *false*. Then the call `subtreeIsSolved`(b,$\mathcal{M}$) updated the formula $b.O$ (line 10). Suppose that `solutionForallChildren` returned *false*, because it found a child node b of node n such that $\mathcal{M}(b.p) = \mathsf{false}$ and `subtreeIsSolved`(b,$\mathcal{M}$) returned *true*. Then the call `subtreeIsSolved`(b,$\mathcal{M}$) updated the formula $b.U$ (line 14). Either way the state has changed, variable L gets a new formula on line 7, and the subsequent call to SMA will produce a different candidate extension.

Example 5. *Apply* `subtreeIsSolved` *to the root r of the* QSMA*-tree $\mathcal{G}$ in Example 2 (Figure 2, left). Formula L gets $r.F = p \vee \neg q$ as $b.U = c.U = \bot$ and $b.O = c.O = \top$.* SMA *produces a model $\mathcal{M}'$ that assigns values to x, p, and q. Suppose that $\mathcal{M}'$ satisfies $p \vee \neg q$ by assigning* true *to p. In the recursive call on b, formula L gets $b.F = \neg F[x, y]$, because b is a leaf. If* SMA *produces an $\mathcal{M}''$ that extends $\mathcal{M}'$ with an assignment to y such that $\mathcal{M}'' \models \neg F[x, y]$, the recursive call on b returns true, which matches the expectation, since $\mathcal{M}'(p) =$* true. *Suppose that $\mathcal{M}'$ satisfies $p \vee \neg q$ by assigning* false *to q. In the recursive call on c, formula L gets $c.F = \neg G[x, z]$, because c is a leaf. If* SMA *fails to produce an $\mathcal{M}''$ that extends $\mathcal{M}'$ with an assignment to z such that*

$\mathcal{M}'' \models \neg G[x,z]$, *the recursive call on* c *returns false, which matches the expectation, since* $\mathcal{M}'(q) = \mathsf{false}$. *Either way, the call on* r *returns true and we have a model of* $\mathcal{G}$.

4.2 Partial Correctness of the QSMA Algorithm

We prove the partial correctness and then the termination of function `subtreeIsSolved`, thus establishing the total correctness of the QSMA algorithm.

Theorem 2 (Partial Correctness)**.** *The function* `subtreeIsSolved` *is partially correct: if the preconditions hold and the function halts, then the postconditions hold.*

Proof. Consider a call `subtreeIsSolved`$(n, \mathcal{M})$, where $\mathcal{M}$ is an extension of $\mathcal{M}_0$ to the variables in $\mathit{Rigid}(n)$. We assume that the preconditions hold and the call terminates, and we show that the postconditions hold (see Algorithm 2 for pre and postconditions). The proof is by structural induction on the tree T_n in the QSMA-tree $\mathcal{G}_n = (T_n, \mathit{Rigid}(n))$ rooted at node n.
Base case: n is a leaf labeled $(\bar{x}, F[\bar{z}, \bar{x}])$. If $\mathcal{M} \models n.U$ and the function returns *true* on line 3 in Algorithm 2 ($rv = \mathit{true}$), we have $\mathcal{M} \models (n.U \vee \neg n.O)$, $\mathcal{M} \not\models \neg n.O$ because $\mathcal{M} \models n.U$ implies $\mathcal{M} \models n.O$, and $\mathcal{M} \models \mathcal{G}_n$ because $\mathcal{M} \models n.U$ implies $\mathcal{M} \models n.\psi$ and n has no children. If $\mathcal{M} \models \neg n.O$ and the function returns *false* on line 5 ($rv = \mathit{false}$), we have $\mathcal{M} \models (n.U \vee \neg n.O)$, $\mathcal{M} \not\models n.U$ as $\mathcal{M} \models \neg n.O$, and $\mathcal{M} \not\models \mathcal{G}_n$ because $\mathcal{M} \models \neg n.O$ implies $\mathcal{M} \not\models n.\psi$. Otherwise, L is assigned $n.F = F[\bar{z}, \bar{x}]$ since n has no children, and SMA is invoked to find an extension $\mathcal{M}'$ of $\mathcal{M}$ to $(FV(n.F) \setminus \mathit{Rigid}(n)) = \bar{x}$ such that $\mathcal{M}' \models n.F$.
If no such extension exists (SMA returned *nil*), $\mathsf{MBO}(n.F, FV(n.F) \setminus \mathit{Rigid}(n), \mathcal{M})$ returns a quantifier-free formula that is false in $\mathcal{M}$, formula $n.O$ is conjoined with this formula, and `subtreeIsSolved` returns *false* on line 11 of Algorithm 2 ($rv = \mathit{false}$). Thus, $\mathcal{M} \not\models n.O$, $\mathcal{M} \models \neg n.O$, $\mathcal{M} \models (n.U \vee \neg n.O)$, $\mathcal{M} \not\models n.U$ as $\mathcal{M} \models \neg n.O$, and $\mathcal{M} \not\models \mathcal{G}_n$ since $\mathcal{M} \models \neg n.O$ (indeed, $n.F = F[\bar{z}, \bar{x}]$, and hence $n.\psi = \exists \bar{x}. F[\bar{z}, \bar{x}]$, could not be satisfied by the call to SMA). If SMA returns an extension $\mathcal{M}'$ of $\mathcal{M}$ to $FV(n.F)$ such that $\mathcal{M}' \models n.F$, `solutionForallChildren`$(n, \mathcal{M}')$ returns *true* because n has no children, L' is assigned $n.F$ for the same reason, $\mathsf{MBU}(n.F, FV(n.F) \setminus \mathit{Rigid}(n), \mathcal{M})$ returns a quantifier-free formula that is true in $\mathcal{M}$, formula $n.U$ is disjoined with this formula, and `subtreeIsSolved` returns *true* on line 15 of Algorithm 2 ($rv = \mathit{true}$). Thus, $\mathcal{M} \models n.U$, $\mathcal{M} \models (n.U \vee \neg n.O)$, $\mathcal{M} \not\models \neg n.O$ because $\mathcal{M} \models n.U$ implies $\mathcal{M} \models n.O$, and $\mathcal{M} \models \mathcal{G}_n$ since $\mathcal{M} \models n.U$ implies $\mathcal{M} \models n.\psi$.
Induction hypothesis: for all children b of node n, if the preconditions are satisfied and `subtreeIsSolved`$(b, \mathcal{M})$ halts, the postconditions are satisfied.
Induction step: if `subtreeIsSolved`$(n, \mathcal{M})$ returns on line 3 or on line 5, the reasoning is the same as in the base case. Otherwise, L is assigned the formula $n.F \wedge \bigwedge_{n \to b}((b.p \Rightarrow b.O) \wedge (\neg b.p \Rightarrow \neg b.U))$. As promised (see Step (2) in the description of `subtreeIsSolved`), we show that the existence of an $\mathcal{M}'$ such that $\mathcal{M}' \models L$ is a necessary condition for $\mathcal{M} \models \mathcal{G}_n$ (†). Indeed, suppose that $\mathcal{M} \models \mathcal{G}_n$. This means that there exists an extension $\mathcal{M}'$ of $\mathcal{M}$ such that: (i) $\mathcal{M}' \models n.F$; (ii) for all children b of n with $\mathcal{M}'(b.p) = \mathsf{true}$, $\mathcal{M}' \models \mathcal{G}_b$ (by Definition 3), so that $\mathcal{M}' \models b.\psi$ (by Theorem 1), and by induction hypothesis ($[\![b.\psi]\!] \subseteq [\![b.O]\!]$) $\mathcal{M}' \models b.O$; and (iii) for all children b of n with $\mathcal{M}'(b.p) = \mathsf{false}$, $\mathcal{M}' \not\models \mathcal{G}_b$ (by Definition 3), so that $\mathcal{M}' \not\models b.\psi$

(by Theorem 1), and by induction hypothesis ($[\![b.U]\!] \subseteq [\![b.\psi]\!]$) $\mathcal{M}' \not\models b.U$, and hence $\mathcal{M}' \models \neg b.U$. By (i), (ii), and (iii), $\mathcal{M}' \models L$.
Function SMA is invoked to find precisely an extension $\mathcal{M}'$ of $\mathcal{M}$ to $FV(L)$ such that $\mathcal{M}' \models L$. If no such extension exists, $\mathsf{MBO}(L, FV(L) \setminus \mathit{Rigid}(n), \mathcal{M})$ returns a quantifier-free formula that is false in $\mathcal{M}$, formula $n.O$ is conjoined with this formula, and the function returns *false* on line 11 ($rv = \mathit{false}$). Therefore, $\mathcal{M} \not\models n.O$, $\mathcal{M} \models \neg n.O$, $\mathcal{M} \models (n.U \vee \neg n.O)$, $\mathcal{M} \not\models n.U$ as $\mathcal{M} \models \neg n.O$ and $\mathcal{M} \not\models \mathcal{G}_n$ by (†), so that the postconditions of `subtreeIsSolved`$(n, \mathcal{M})$ are satisfied. If there exists an extension $\mathcal{M}'$ such that $\mathcal{M}' \models L$, `solutionForallChildren`$(n, \mathcal{M}')$ is invoked. If it returns *true*, L' is assigned the formula $n.F \wedge \bigwedge_{n \rightarrow b}((b.p \Rightarrow b.U) \wedge (\neg b.p \Rightarrow \neg b.O))$. As promised (see Step (6) in the description of `subtreeIsSolved`), we show that (I) $\mathcal{M}' \models L'$ and (II) $\mathcal{M}' \models L'$ is a sufficient condition for $\mathcal{M} \models \mathcal{G}_n$.
For (I), $\mathcal{M}' \models n.F$, because $\mathcal{M}' \models L$, and from the knowledge that $\mathcal{M} \models \mathcal{G}_n$ (`solutionForallChildren` returned *true*) we know that for all children b of n, if $\mathcal{M}'(b.p) = \mathsf{true}$, `subtreeIsSolved`$(b, \mathcal{M}')$ returned *true*, so that $\mathcal{M}' \models b.U$ by induction hypothesis, and if $\mathcal{M}'(b.p) = \mathsf{false}$, `subtreeIsSolved`$(b, \mathcal{M}')$ returned *false*, so that $\mathcal{M}' \models \neg b.O$ by induction hypothesis.
For (II), $\mathcal{M}' \models L'$ implies (i) $\mathcal{M}' \models n.F$, and (ii) for all children b of n: if $\mathcal{M}'(b.p) = \mathsf{true}$, then $\mathcal{M}' \models L'$ implies $\mathcal{M}' \models b.U$, and by induction hypothesis $\mathcal{M}' \models \mathcal{G}_b$; if $\mathcal{M}'(b.p) = \mathsf{false}$, then $\mathcal{M}' \models L'$ implies $\mathcal{M}' \models \neg b.O$, and by induction hypothesis $\mathcal{M}' \not\models \mathcal{G}_b$. Thus, $\mathcal{M} \models \mathcal{G}_n$ by Definition 3.
Then, $\mathsf{MBU}(L', FV(L') \setminus \mathit{Rigid}(n), \mathcal{M})$ returns a quantifier-free formula that is true in $\mathcal{M}$, formula $n.U$ is disjoined with this formula, and the function returns *true* on line 15 ($rv = \mathit{true}$). Therefore, $\mathcal{M} \models n.U$, $\mathcal{M} \models (n.U \vee \neg n.O)$, $\mathcal{M} \not\models \neg n.O$ because $\mathcal{M} \models n.U$ implies $\mathcal{M} \models n.O$, and $\mathcal{M} \models \mathcal{G}_n$ by (II), so that the postconditions are satisfied. □

4.3 Termination of the QSMA Algorithm

For termination, we begin by considering the role of the MBU and MBO functions. Suppose that $\mathcal{T}$ is LRA and let Q denote the sort of the rationals and $\mathbb{Q}$ the set of the rational numbers. The theory extension LRA^+ adds constant symbols $\tilde{q}$ of sort Q for all rational numbers $q \in \mathbb{Q}$. An MBU function such that $\mathsf{MBU}(F[\bar{z}, x], x, \mathcal{M}) = F[\bar{z}, x]\{x \leftarrow \tilde{q}\}$, and $\mathcal{M} \models F[\bar{z}, x]\{x \leftarrow \tilde{q}\}$, produces an under-approximation of $\exists x.F[\bar{z}, x]$, but it is not a good choice for termination, since the domain $\mathbb{Q}$ is infinite. If the QSMA algorithm were to apply this MBU repeatedly with an enumeration of rational constants, it could build an infinite sequence of under-approximations $(\bigvee_{i=1}^{n} F[\bar{z}, x]\{x \leftarrow \tilde{q}_i\})_{n \in \mathbb{N}}$, none of which is LRA-equivalent to $\exists x.F[\bar{z}, x]$. The next definition excludes such MBU functions, by requiring that for a given formula MBU generates only finitely many formulas.

Definition 6 (MBU with finite basis)**.** *An* MBU *function has* finite basis *if the set* $\{\mathsf{MBU}(F[\bar{z}, \bar{x}], \bar{x}, \mathcal{M}) \mid \mathcal{M} : \textit{extension of } \mathcal{M}_0 \textit{ to } \bar{z} \textit{ such that } \mathcal{M} \models \exists \bar{x}.F[\bar{z}, \bar{x}]\}$ *is finite for all quantifier-free formulas* $F[\bar{z}, \bar{x}]$ *and tuples* $\bar{x}$*.*

The notion of an MBO function having a finite basis is defined analogously.

Definition 7 (MBO with finite basis)**.** *An* MBO *function has* finite basis *if the set* $\{\mathsf{MBO}(F[\bar{z},\bar{x}],\bar{x},\mathcal{M}) \mid \mathcal{M}$: *extension of* $\mathcal{M}_0$ *to* $\bar{z}$ *such that* $\mathcal{M} \not\models \exists\bar{x}.F[\bar{z},\bar{x}]\}$ *is finite for all quantifier-free formulas* $F[\bar{z},\bar{x}]$ *and tuples* $\bar{x}$*.*

The following lemma proves a key consequence of the finite basis assumption that will be used in the termination theorem.

Lemma 3. *If* MBU *and* MBO *have finite basis, for all (possibly infinite) series of calls* $\{\texttt{subtreeIsSolved}(n,\mathcal{M}_i)\}_i$*, all satisfying the preconditions and all terminating, formulas* $n.U$ *and* $n.O$ *are updated only a finite number of times.*

Proof. The proof is by structural induction on the tree T_n in the QSMA-tree $\mathcal{G}_n = (T_n, \mathit{Rigid}(n))$ rooted at node n. Given a series of calls as in the claim, let $(n.U)_i$ and $(n.O)_i$ denote the values of $n.U$ and $n.O$ upon entering call $\texttt{subtreeIsSolved}(n,\mathcal{M}_i)$.
Base case: n is a leaf labeled $(\bar{x}, F[\bar{z},\bar{x}])$. We consider first $n.O$ and then $n.U$.
For $n.O$, for all indices i in the series of calls, either (I) $(n.O)_{i+1} = (n.O)_i$ or (II) $(n.O)_{i+1} = (n.O)_i \wedge \mathsf{MBO}(L_i, FV(L_i) \setminus \mathit{Rigid}(n), \mathcal{M}_i)$ where $L_i = n.F = F[\bar{z},\bar{x}]$ and $(FV(L_i) \setminus \mathit{Rigid}(n)) = \bar{x}$, because n is a leaf. Case (II) applies only if $\mathcal{M}_i \models (n.O)_i$ (if we enter the main loop $\mathcal{M}_i \models \neg(n.U)_i \wedge (n.O)_i$), $\mathcal{M}_i \models \neg(n.O)_{i+1}$, and $\texttt{subtreeIsSolved}(n,\mathcal{M}_i)$ returns *false* (see lines 10-11 in Algorithm 2 and Step (3) in the description of $\texttt{subtreeIsSolved}$). Since for all i, $L_i = n.F$ and $(FV(L_i) \setminus \mathit{Rigid}(n)) = \bar{x}$, whenever we hit Case (II), MBO is applied to the same formula and variable tuple, while the third argument (the extension) may vary. By the finite basis hypothesis, MBO can generate only finitely many formulas for a given formula and variable tuple. Thus, $n.O$ can be updated only a finite number of times.
For $n.U$, for all indices i in the series of calls, either (I) $(n.U)_{i+1} = (n.U)_i$ or (II) $(n.U)_{i+1} = (n.U)_i \vee \mathsf{MBU}(L'_i, FV(L'_i) \setminus \mathit{Rigid}(n), \mathcal{M}_i)$ where as before $L'_i = n.F$ and $(FV(L_i) \setminus \mathit{Rigid}(n)) = \bar{x}$. Case (II) applies only if $\mathcal{M}_i \models \neg(n.U)_i$, $\mathcal{M}_i \models (n.U)_{i+1}$, and $\texttt{subtreeIsSolved}(n,\mathcal{M}_i)$ returns *true* (see lines 14-15 in Algorithm 2 and Step (6) in the description of $\texttt{subtreeIsSolved}$). Since for all i, $L'_i = n.F$ and $(FV(L_i) \setminus \mathit{Rigid}(n)) = \bar{x}$, whenever we hit Case (II), MBU is applied to the same formula and variable tuple. By the finite basis hypothesis, $n.U$ can be updated only a finite number of times.
Induction hypothesis: the claim holds for all children b of n.
Induction step: given a series of calls as in the claim, for all children b of n, let $(b.U)_i$ and $(b.O)_i$ denote the values of $b.U$ and $b.O$ upon entering call $\texttt{subtreeIsSolved}(n,\mathcal{M}_i)$. By induction hypothesis, for all children b of n, $b.U$ and $b.O$ are updated only a finite number of times. Therefore, there exists an index i_0 in the series of calls, such that for all $i \geq i_0$, for all children b of n, $(b.U)_{i+1} = (b.U)_i$ and $(b.O)_{i+1} = (b.O)_i$. We consider first $n.O$ and then $n.U$.
For $n.O$, we have that for all i, $i \geq i_0$, either (I) $(n.O)_{i+1} = (n.O)_i$ or (II) $(n.O)_{i+1} = (n.O)_i \wedge \mathsf{MBO}(L_i, FV(L_i) \setminus \mathit{Rigid}(n), \mathcal{M}_i)$ where

$$L_i = n.F \wedge \bigwedge_{n \to b} ((b.p \Rightarrow (b.O)_i) \wedge (\neg b.p \Rightarrow \neg(b.U)_i)).$$

Since for all i, $i \geq i_0$, for all children b of n, $(b.U)_{i+1} = (b.U)_i$ and $(b.O)_{i+1} = (b.O)_i$, it follows that for all i, $i \geq i_0$, $L_{i+1} = L_i$. Therefore, for all i, $i \geq i_0$, whenever we hit Case (II), MBO is applied to the same formula and variable tuple. By the finite basis hypothesis, $n.O$ can be updated only a finite number of times.
For $n.U$, for all i, $i \geq i_0$, either (I) $(n.U)_{i+1} = (n.U)_i$ or (II) $(n.U)_{i+1} = (n.U)_i \vee \mathsf{MBU}(L'_i, FV(L'_i) \setminus \mathit{Rigid}(n), \mathcal{M}_i)$ where

$$L'_i = n.F \wedge \bigwedge_{n \to b} ((b.p \Rightarrow (b.U)_i) \wedge (\neg b.p \Rightarrow \neg(b.O)_i)).$$

Since for all i, $i \geq i_0$, for all children b of n, $(b.U)_{i+1} = (b.U)_i$ and $(b.O)_{i+1} = (b.O)_i$, it follows that for all i, $i \geq i_0$, $L'_{i+1} = L'_i$. Therefore, for all i, $i \geq i_0$, whenever we hit Case (II), MBU is applied to the same formula and variable tuple. By the finite basis hypothesis, $n.U$ can be updated only a finite number of times. □

Once nontermination due to MBU or MBO is excluded even for an infinite series of halting calls, termination is proved by induction on the QSMA-tree.

Theorem 4 (Termination)**.** *If the* MBU *and* MBO *functions have finite basis, whenever the preconditions are satisfied the function* `subtreeIsSolved` *halts.*

Proof. Consider a call `subtreeIsSolved`$(n, \mathcal{M})$, where $\mathcal{M}$ is an extension of $\mathcal{M}_0$ to the variables in $\mathit{Rigid}(n)$. We assume that the preconditions hold (see Algorithm 2) and we show that the call terminates. The proof is by structural induction on the tree T_n in the QSMA-tree $\mathcal{G}_n = (T_n, \mathit{Rigid}(n))$ rooted at node n.
Base case: if n is a leaf there is no recursion and the call terminates.
Induction hypothesis: the claim holds for all children $b_1, \ldots, b_k$ of n.
Induction step: if `subtreeIsSolved`$(n, \mathcal{M})$ does not enter the main loop, it halts. Suppose that it enters the main loop. For this case we reason by way of contradiction, assuming that `subtreeIsSolved`$(n, \mathcal{M})$ does not halt. This means that the SMA function produces an infinite series of candidate models $\{\mathcal{M}_i\}_{i \geq 1}$ such that for all i, $i \geq 1$, there exists a child $b_{j(i)}$, $1 \leq j(i) \leq k$, for which $\mathcal{M}_i(b_{j(i)}.p) \neq$ `subtreeIsSolved`$(b_{j(i)}, \mathcal{M}_i)$ so that `solutionForallChildren` returns *false* (lines 19-20 in Algorithm 2). It follows that `subtreeIsSolved`$(n, \mathcal{M})$ generates an infinite series $\mathcal{S}$ of recursive calls.
Let W be a matrix with a row for each M_i, $i \geq 1$, a column for each b_h, $1 \leq h \leq k$, and such that $W_{i,h} = 1$ if $\mathcal{M}_i(b_h.p) =$ `subtreeIsSolved`$(b_h, \mathcal{M}_i)$, $W_{i,h} = 0$ if $\mathcal{M}_i(b_h.p) \neq$ `subtreeIsSolved`$(b_h, \mathcal{M}_i)$, and $W_{i,h} = \perp$ if `subtreeIsSolved` is not invoked on $(b_h, \mathcal{M}_i)$. Recall that the latter case may happen, because `solutionForallChildren` returns *false* as soon as it finds a child node for which `subtreeIsSolved` returns a Boolean value other than the expected one.
For all h, $1 \leq h \leq k$, let $D_h = \{i \mid W_{i,h} = 0\}$. In words, D_h is the subset of indices of those candidate models in the $\{\mathcal{M}_i\}_{i \geq 1}$ series, for which child node b_h is the one that causes `solutionForallChildren` to return *false*. By projecting on the node argument, we extract from $\mathcal{S}$ up to k (possibly infinite) series of calls $\{$`subtreeIsSolved`$(b_h, \mathcal{M}_i)\}_{i \in D_h}$. Consider anyone of these series and let us temporarily rename b_h as b for simplicity. For all the calls `subtreeIsSolved`$(b, \mathcal{M}_i)$ in the

series, since $\mathcal{M}_i$ was produced by SMA (line 8 in Algorithm 2), we know that $\mathcal{M}_i \models L$. In other words, we know that before the call `subtreeIsSolved`$(b, \mathcal{M}_i)$ it holds that

$$\mathcal{M}_i \models (b.p \Rightarrow b.O) \wedge (\neg b.p \Rightarrow \neg b.U).$$

If $\mathcal{M}_i(b.p) = \mathsf{true}$, then before the call $\mathcal{M}_i \models b.O$. Since the call returns *false*, it means that the call has updated $b.O$ to ensure that $\mathcal{M}_i \models \neg b.O$ (line 10 in Algorithm 2 and Step (3) in the description of `subtreeIsSolved`). Similarly, if $\mathcal{M}_i(b.p) = \mathsf{false}$, then before the call $\mathcal{M}_i \models \neg b.U$. Since the call returns *true*, it means that the call has updated $b.U$ to ensure that $\mathcal{M}_i \models b.U$ (line 14 in Algorithm 2 and Step (6) in the description of `subtreeIsSolved`). In summary, at least one of $b.U$ or $b.O$ gets updated for each call in the series. However, by induction hypothesis all the calls in all the possibly infinite series $\{$`subtreeIsSolved`$(b_h, \mathcal{M}_i)\}_{i \in D_h}$ terminate. Therefore, Lemma 3 applies to each of these series, establishing that $b_h.U$ and $b_h.O$ get updated only a finite number of times. Therefore, all the series $\{$`subtreeIsSolved`$(b_h, \mathcal{M}_i)\}_{i \in D_h}$ are finite, which contradicts the existence of the infinite series $\mathcal{S}$. □

Example 6. *Apply* `subtreeIsSolved` *to the root r of the* QSMA*-tree $\mathcal{G}$ for formula φ in Example 3 (Figure 2, right). Formula L gets $p \wedge \neg q$, because $b.U = c.U = \bot$ and $b.O = c.O = \top$.* SMA *produces an extension $\mathcal{M}'$ that assigns values to x, p, and q. Suppose that $\mathcal{M}'$ assigns 1 to x, while it must assign* true *to p and* false *to q. In the recursive call on node b, formula L gets $x \simeq 2{\cdot}y$, because b is a leaf. If* SMA *produces an $\mathcal{M}''$ that extends $\mathcal{M}'$ with $y{\leftarrow}\frac{1}{2}$, the recursive call on node b returns true, which matches the expectation, since $\mathcal{M}'(p) = $* true*, and we have a model of $\mathcal{G}_b$. In the recursive call on node c, formula L gets $3{\cdot}x \simeq 2{\cdot}z$, because c is a leaf. If* SMA *produces an $\mathcal{M}''$ that extends $\mathcal{M}'$ with $z{\leftarrow}\frac{3}{2}$, the recursive call on node c returns true, and we have a model of $\mathcal{G}_c$, but because $\mathcal{M}'(q) = $* false*, the expected truth value is not produced. Therefore, the call on r returns false, which means there is no model of $\mathcal{G}$. Indeed, formula φ of Example 3 is false in* LRA *as the original formula $\neg\varphi$ is true in* LRA.

5 The **OptiQSMA** Algorithm and Its Correctness

In this section we present OptiQSMA, the optimized variant of QSMA implemented in the YicesQS solver. OptiQSMA reduces the number of recursive calls to the `subtreeIsSolved` function, by entrusting more work to each call to the SMA function. Reconsider the behavior of QSMA in Example 5. We can avoid a recursive call to `subtreeIsSolved` by asking SMA to satisfy $(p \vee \neg q) \wedge (p \Rightarrow \neg F[x, y])$ in lieu of $p \vee \neg q$. This way, if the candidate model returned by SMA assigns true to p, it also assigns to x and y values that satisfy $\neg F[x, y]$. This means that $\exists y. \neg F[x, y]$ is found true without recursion. On the other hand, if q is assigned false, the algorithm still has to make the recursive call to see if it can satisfy $\exists z. \neg G[x, z]$.

5.1 The Look-ahead Optimization

In general, the idea of OptiQSMA is to do a look-ahead on a path in the QSMA-tree, doing the work in one shot rather than through recursive calls on all the nodes in the path. The look-ahead applies to a path such that the Boolean labels of all the arcs in the path are assigned true by the candidate model. The following definition builds a formula to allow the look-ahead. Recall that a conjunction indexed by $n \to b$ means the conjunction of the stated formula for all children nodes b of node n.

Definition 8 (Look-ahead formula)**.** *Given a* QSMA*-tree* $\mathcal{G} = (\bar{z}, T)$*, for all nodes* n *of* T *the* look-ahead formula *of* n *is* $LF(n) = n.F \wedge \bigwedge_{n \to b}(b.p \Rightarrow LF(b))$.

The look-ahead formula of a node n captures the whole subtree rooted at n. We still need to distinguish between the nodes that are handled together in one shot without recursion and those where recursion is needed. This is accomplished by the next two definitions. The first one defines the *no-alternation nodes*, which are on a path where all labels are assigned true. The second one introduces the *first-alternation nodes*, namely those reached by the first arc whose label is assigned false.

Definition 9 (No-alternation nodes)**.** *Given a* QSMA*-tree* $\mathcal{G} = (\bar{z}, T)$ *for all nodes* n *of* T *and extensions* $\mathcal{M}$ *of* $\mathcal{M}_0$ *to* $FV(LF(n))$*, the set* NAN$(n, \mathcal{M})$ *of the* no-alternation nodes from n according to $\mathcal{M}$ *contains all and only the nodes* b *such that: (i)* b *is a descendant of* n *through a path* $n \to n_1 \to \ldots \to n_q \to b$ $(q \geq 0)$*, (ii)* $\forall i$, $1 \leq i \leq q$, $\mathcal{M}(n_i.p) = \mathsf{true}$*, and (iii)* $\mathcal{M}(b.p) = \mathsf{true}$.

Definition 10 (First-alternation nodes)**.** *Given a* QSMA*-tree* $\mathcal{G} = (\bar{z}, T)$ *for all nodes* n *of* T *and extensions* $\mathcal{M}$ *of* $\mathcal{M}_0$ *to* $FV(LF(n))$*, the set* FAN$(n, \mathcal{M})$ *of the* first-alternation nodes from n according to $\mathcal{M}$ *contains all and only the nodes* b *such that: (i)* b *is a descendant of* n *through a path* $n \to n_1 \to \ldots \to n_q \to b$ $(q \geq 0)$*, (ii)* $\forall i$, $1 \leq i \leq q$, $\mathcal{M}(n_i.p) = \mathsf{true}$*, and (iii)* $\mathcal{M}(b.p) = \mathsf{false}$.

A node $b \in$ FAN$(n, \mathcal{M})$ such that $q = 0$ in Condition (i) of Definition 10 is a child of n: for a child there is no optimization. The OptiQSMA algorithm seeks a candidate model $\mathcal{M}$ that satisfies $LF(n)$ and recurses only on the nodes in FAN$(n, \mathcal{M})$. Therefore, the definition of *satisfaction with look-ahead*, denoted $\models_{la}$, follows the pattern of Definition 3, replacing $r.F$ with $LF(r)$ and Condition (ii) of Definition 3 with a condition for the first alternation nodes.

Definition 11 (Satisfaction with look-ahead)**.** *Given a* QSMA*-tree* $\mathcal{G} = (\bar{z}, T)$ *with* $r = root(T)$ *and an extension* $\mathcal{M}$ *of* $\mathcal{M}_0$ *to the set* $Rigid(r) = \bar{z}$ *of the rigid variables at node* r*,* $\mathcal{M} \models_{la} \mathcal{G}$ *if there exists an extension* $\mathcal{M}'$ *of* $\mathcal{M}$ *to* $FV(LF(r)) \setminus Rigid(r)$ *such that (i)* $\mathcal{M}' \models LF(r)$ *and (ii)* $\mathcal{M}' \not\models_{la} \mathcal{G}_b$ *for all nodes* $b \in$ FAN$(r, \mathcal{M}')$.

Since $\mathcal{M}'(b.p) = \mathsf{false}$ for the nodes $b \in$ FAN$(r, \mathcal{M}')$, the $\models_{la}$ relation is negated in Condition (ii) of Definition 11. The next theorem shows that the look-ahead optimization does not change the problem.

Theorem 5 (Equivalence of satisfaction and satisfaction with look-ahead)**.** *Given a* QSMA*-tree* $\mathcal{G} = (\bar{z}, T)$ *and an extension* $\mathcal{M}$ *of* $\mathcal{M}_0$ *to* $\bar{z}$*,* $\mathcal{M} \models \mathcal{G}$ *iff* $\mathcal{M} \models_{la} \mathcal{G}$.

Proof. The proof is by structural induction on the tree T. Let $r = root(T)$.
Base case: if root r is the only node in T, the claim holds, because $LF(r) = r.F$, $FV(LF(r)) \setminus Rigid(r) = FV(r.F) \setminus Rigid(r) = Var(r)$, and both Condition (ii) in Definition 3 and Condition (ii) in Definition 11 are vacuously true.

Induction hypothesis: for all children nodes b of node r the claim holds.
Induction step: we distinguish the two directions.
$\Rightarrow$) By hypothesis, $\mathcal{M} \models \mathcal{G}$, that is, there exists an extension $\mathcal{M}'$ of $\mathcal{M}$ to $\mathit{Var}(r)$ that fulfills Definition 3. We build an extension $\mathcal{M}''$ of $\mathcal{M}'$ to $FV(LF(r)) \setminus \mathit{Rigid}(r)$ that fits Definition 11. First, $FV(LF(r)) = FV(r.F) \cup \{b.p \mid r \rightarrow b\} \cup \bigcup_{r \rightarrow b} FV(LF(b))$. Note that $FV(r.F) \subseteq \mathit{Rigid}(r) \cup \mathit{Var}(r)$ and $\{b.p \mid r \rightarrow b\} \subseteq \mathit{Var}(r)$. Since $\mathcal{M}$ interprets the variables in $\mathit{Rigid}(r)$ and $\mathcal{M}'$ extends $\mathcal{M}$ to interpret the variables in $\mathit{Var}(r)$, we need to consider only the variables in $(\bigcup_{r \rightarrow b} FV(LF(b))) \setminus \mathit{Rigid}(r)$. Since $FV(LF(b))$ may contain variables that are in $\mathit{Rigid}(b) = \mathit{Rigid}(r) \cup \{\bar{x}\}$ for $\bar{x}$ the local variables of r and $\bar{x} \subseteq \mathit{Var}(r)$, $\mathcal{M}''$ only needs to add interpretations of the variables in $FV(LF(b)) \setminus \mathit{Rigid}(b)$ for all children nodes b of node r.
Let b be a child node of node r such that $\mathcal{M}'(b.p) = \mathsf{false}$. Then, for all $y \in FV(LF(b)) \setminus \mathit{Rigid}(b)$, let $\mathcal{M}''$ assign an arbitrary value to y.
Let b be a child node of node r such that $\mathcal{M}'(b.p) = \mathsf{true}$. Since $\mathcal{M} \models \mathcal{G}$, by Definition 3, $\mathcal{M}' \models \mathcal{G}_b$, and by induction hypothesis $\mathcal{M}' \models_{la} \mathcal{G}_b$, that is, there exists an extension $\mathcal{M}'_b$ of $\mathcal{M}'$ fulfilling Definition 11 for $\mathcal{G}_b$. Then, for all $y \in FV(LF(b)) \setminus \mathit{Rigid}(b)$, let $\mathcal{M}''(y) = \mathcal{M}'_b(y)$ (†).
This construction of $\mathcal{M}''$ does not assign two different values to the same variable, because if b and b' are distinct children of r, we have

$$FV(LF(b)) \cap FV(LF(b')) \subseteq \mathit{Rigid}(b) = \mathit{Rigid}(b').$$

We show that $\mathcal{M}''$ fulfills Condition (i) in Definition 11. First, $\mathcal{M}' \models r.F$ implies $\mathcal{M}'' \models r.F$, since $FV(r.F) \subseteq \mathit{Rigid}(r) \cup \mathit{Var}(r)$. Second, for all children b of r, $b.p \in \mathit{Var}(r)$ and hence $\mathcal{M}''(b.p) = \mathcal{M}'(b.p)$. For all children b of r such that $\mathcal{M}''(b.p) = \mathcal{M}'(b.p) = \mathsf{false}$, the conjunct for b in $LF(r)$ is vacuously true. For all children b of r such that $\mathcal{M}''(b.p) = \mathcal{M}'(b.p) = \mathsf{true}$, we know that $\mathcal{M}'_b \models LF(b)$ and hence $\mathcal{M}'' \models LF(b)$ by (†). Therefore, $\mathcal{M}'' \models LF(r)$.
We show that $\mathcal{M}''$ fulfills Condition (ii) in Definition 11. Let $b \in \mathsf{FAN}(r, \mathcal{M}'')$ be a descendant of r via a path $r \rightarrow n_1 \rightarrow \ldots \rightarrow n_q \rightarrow b$. If $q = 0$, node b is a child of node r, and $\mathcal{M}''(b.p) = \mathcal{M}'(b.p) = \mathsf{false}$. Since $\mathcal{M} \models \mathcal{G}$ with extension $\mathcal{M}'$, we have that $\mathcal{M}' \not\models \mathcal{G}_b$. By induction hypothesis, $\mathcal{M}' \not\models_{la} \mathcal{G}_b$. Since $\mathcal{M}''$ is an extension of $\mathcal{M}'$, also $\mathcal{M}'' \not\models_{la} \mathcal{G}_b$ holds. If $q > 0$, node n_1 is a child of node r, and $\mathcal{M}''(n_1.p) = \mathcal{M}'(n_1.p) = \mathsf{true}$. Since $\mathcal{M} \models \mathcal{G}$ with extension $\mathcal{M}'$, we have that $\mathcal{M}' \models \mathcal{G}_{n_1}$. By induction hypothesis, $\mathcal{M}' \models_{la} \mathcal{G}_{n_1}$ with some extension $\mathcal{M}'_{n_1}$. Since $b \in \mathsf{FAN}(n_1, \mathcal{M}'_{n_1})$, by Definition 11 applied to n_1, we have that $\mathcal{M}'_{n_1} \not\models_{la} \mathcal{G}_b$. By (†), $\mathcal{M}''$ is an extension of $\mathcal{M}'$ that interprets all the variables in $FV(LF(n_1)) \setminus \mathit{Rigid}(n_1)$ like $\mathcal{M}'_{n_1}$ does. Thus, also $\mathcal{M}'' \not\models_{la} \mathcal{G}_b$ holds as desired.
$\Leftarrow$) By hypothesis, $\mathcal{M} \models_{la} \mathcal{G}$, that is, there exists an extension $\mathcal{M}'$ of $\mathcal{M}$ to $FV(LF(r)) \setminus \mathit{Rigid}(r)$ that fulfills Definition 11. We show that $\mathcal{M}'$ itself fulfills the requirements in Definition 3. Condition (i) in Definition 3 is satisfied, because $\mathcal{M}' \models LF(r)$ implies $\mathcal{M}' \models r.F$. For Condition (ii) in Definition 3, we reason as follows. For all children b of node r such that $\mathcal{M}'(b.p) = \mathsf{false}$, $b \in \mathsf{FAN}(r, \mathcal{M}')$, and hence by Definition 11, $\mathcal{M}' \not\models_{la} \mathcal{G}_b$. By induction hypothesis, $\mathcal{M}' \not\models \mathcal{G}_b$. For all children b of node r such that $\mathcal{M}'(b.p) = \mathsf{true}$, $\mathcal{M}' \models LF(r)$ implies $\mathcal{M}' \models LF(b)$, so that Condition (i) in Definition 11 applied to node b is satisfied. Since $\mathsf{FAN}(b, \mathcal{M}') \subseteq \mathsf{FAN}(r, \mathcal{M}')$,

Condition (ii) in Definition 11 applied to node b is satisfied by hypothesis. Thus, $\mathcal{M}' \models_{la} \mathcal{G}_b$. By induction hypothesis, $\mathcal{M}' \models \mathcal{G}_b$. Therefore, $\mathcal{M}'$ fulfills Definition 3 and $\mathcal{M} \models \mathcal{G}$. □

5.2 Description of the **OptiQSMA** Algorithm

Algorithm 3 Pseudocode of the main function of OptiQSMA

```
@pre: $\mathcal{G} = (\bar{z}, T)$: QSMA-tree for $\varphi$ with $FV(\varphi) = \bar{z}$; $\mathcal{M}$: extension of $\mathcal{M}_0$ to $\bar{z}$
@post: $rv$ iff $\mathcal{M} \models_{la} \mathcal{G}$
1: function OptiQSMA($\mathcal{M}$, $T$)
2:     for all nodes $n$ in $T$ do
3:         $n.U \leftarrow \bot$
4:     ans $\leftarrow$ optiSubtreeIsSolved($root(T)$, $\mathcal{M}$)
5:     if ans = SAT(_) then
6:         return $true$
7:     if ans = UNSAT(_) then
8:         return $false$
```

Let $\mathcal{G} = (\bar{z}, T)$ be the QSMA-tree for input formula φ with $FV(\varphi) = \bar{z}$. Given a model $\mathcal{M}$ extending $\mathcal{M}_0$ to the variables in $\bar{z}$, the OptiQSMA algorithm determines whether $\mathcal{M} \models_{la} \mathcal{G}$. Another optimization is that OptiQSMA keeps for all nodes n in T only under-approximations $n.U$ of $n.\psi$ (the formula at node n). OptiQSMA also computes under-approximations U and over-approximations O, weakening under-approximations and strengthening over-approximations, in order to inflate the $[\![U]\!]$ bubble and deflate the $[\![O]\!]$ bubble (Figure 3), but over-approximations are computed and used, not saved. Accordingly, the main function OptiQSMA (Algorithm 3) only initializes $n.U$ for all nodes n.

Then OptiQSMA calls function `optiSubtreeIsSolved` (Algorithm 4) with arguments $root(T)$ and $\mathcal{M}$. This call to function `optiSubtreeIsSolved` returns SAT(U) if $\mathcal{M} \models_{la} \mathcal{G}$ and UNSAT(O) if $\mathcal{M} \not\models_{la} \mathcal{G}$. Formula U is an under-approximation of $r.\psi$ ($r = root(T)$) such that $\mathcal{M} \models U$ (and hence $\mathcal{M} \models \mathcal{G}$ so that $\mathcal{M} \models_{la} \mathcal{G}$ by Theorem 5). Formula O is an over-approximation of $r.\psi$ such that $\mathcal{M} \not\models O$ (and hence $\mathcal{M} \not\models \mathcal{G}$ so that $\mathcal{M} \not\models_{la} \mathcal{G}$ by Theorem 5). The main function OptiQSMA has no usage for formulas U and O and merely returns $true$ or $false$ accordingly.

Function `optiSubtreeIsSolved` takes a node n and a model $\mathcal{M}$ extending $\mathcal{M}_0$ to $Rigid(n)$ (precondition of Algorithm 4), and maintains the invariant $\forall b \in T.\ [\![b.U]\!] \subseteq [\![b.\psi]\!]$ (both pre and postcondition), which only refers to under-approximations because over-approximations are not kept. Function `optiSubtreeIsSolved` returns SAT(U) if $\mathcal{M} \models_{la} \mathcal{G}_n$ and UNSAT(O) if $\mathcal{M} \not\models_{la} \mathcal{G}_n$ (postcondition). Formulas U and O are generated by invoking the model-based approximation functions MBU and MBO. While OptiQSMA has no usage for the returned formulas U and O, function `optiSubtreeIsSolved` itself uses them, because it works recursively, building and returning under- and over-approximations. The reason

Algorithm 4 Pseudocode of the auxiliary functions of OptiQSMA

```
@pre: $\mathcal{M}$: extension of $\mathcal{M}_0$ to $Rigid(n)$ and $\forall b \in T.\ [\![b.U]\!] \subseteq [\![b.\psi]\!]$
@post: $\forall b \in T.\ [\![b.U]\!] \subseteq [\![b.\psi]\!]$ and
$\{rv = \mathsf{UNSAT}(O)$ implies $[(\forall b \in T.\ [\![b.\psi]\!] \subseteq [\![O]\!])$ and $\mathcal{M} \not\models O]\}$ and
$\{rv = \mathsf{SAT}(U)$ implies $[(\forall b \in T.\ [\![b.U]\!] \subseteq [\![b.\psi]\!])$ and $\mathcal{M} \models U]\}$
 1: function optiSubtreeIsSolved($n$, $\mathcal{M}$)
 2:     while true do
 3:         $L \leftarrow LF(n) \wedge \bigwedge_{n \rightarrow^+ b}(\neg b.p \Rightarrow \neg b.U)$
 4:         $\mathcal{M}' \leftarrow \mathsf{SMA}(L, \mathcal{M})$
 5:         if $\mathcal{M}' = nil$ then
 6:             return $\mathsf{UNSAT}(\mathsf{MBO}(L, FV(L) \setminus Rigid(n), \mathcal{M}))$
 7:         reasons $\leftarrow \top$
 8:         if solutionForallDescendants($n$, $\mathcal{M}'$, reasons) then
 9:             $L' \leftarrow LF(n) \wedge$ reasons
10:             return $\mathsf{SAT}(\mathsf{MBU}(L', FV(L') \setminus Rigid(n), \mathcal{M}))$
11:
12: function solutionForallDescendants($n$, $\mathcal{M}$, reasons)
13:     for all $b \in \mathsf{FAN}(n, \mathcal{M})$ do
14:         ans $\leftarrow$ optiSubtreeIsSolved($b$, $\mathcal{M}$)
15:         if ans $= \mathsf{SAT}(U)$ then
16:             $b.U \leftarrow b.U \vee U$
17:             return $false$
18:         if ans $= \mathsf{UNSAT}(O)$ then
19:             reasons $\leftarrow$ reasons $\wedge (\neg b.p \Rightarrow \neg O)$
20:     for all $b \in \mathsf{NAN}(n, \mathcal{M})$ do
21:         reasons $\leftarrow$ reasons $\wedge\ b.p$
22:     return $true$
```

for saving only under-approximations is practical. It is elucidated in the illustration of `optisubtreeIsSolved` and its proof of partial correctness. Function `optisubtreeIsSolved` executes a loop whose body contains the following steps:

1. Build a formula L (line 3 in Algorithm 4) as the conjunction of the look-ahead formula $LF(n)$ (in lieu of $n.F$ in line 7 of Algorithm 2) and a formula for every descendant node b of n, denoted $n \rightarrow^+ b$ (in lieu of child node as in Algorithm 2).
2. Invoke the SMA function to search for an extension $\mathcal{M}'$ of $\mathcal{M}$ to $Var(n)$ such that $\mathcal{M}' \models L$ (line 4). For those descendants b for which $\mathcal{M}'(b.p) = \mathsf{false}$, the subformula for b in L reduces to $\neg b.U$ as in Step (2) of the description of `subtreeIsSolved`. For those descendants b for which $\mathcal{M}'(b.p) = \mathsf{true}$, the subformula for b in L reduces to true, in agreement with the fact that over-approximations are not kept.
3. If SMA returns nil, function `optiSubtreeIsSolved` returns $\mathsf{UNSAT}(O)$, where O is directly the outcome of applying MBO to L and $\mathcal{M}$, since over-approximations are not kept (line 6). Otherwise (i.e., SMA returns an extension $\mathcal{M}'$), there is potential for satisfaction with look-ahead. Function `optiSubtreeIsSolved` initializes the formula `reasons` to $\top$. Then it invokes `solutionForallDescendants` passing

extension $\mathcal{M}'$ and node n by value (as in the call to `solutionForallChildren` by `subtreeIsSolved` in Algorithm 2) and formula `reasons` by reference.

4. Function `solutionForallDescendants` considers first all descendant nodes b in $\mathsf{FAN}(n, \mathcal{M})$, and calls `optiSubtreeIsSolved`$(b, \mathcal{M})$ for each of them. If the call returns $\mathsf{SAT}(U)$, it means that $\mathcal{M} \models_{la} \mathcal{G}_b$. Since $b \in \mathsf{FAN}(n, \mathcal{M})$ means that $\mathcal{M}(b.p) = \mathsf{false}$ (see Definition 10), there is a discrepancy with respect to the expected truth value, and hence function `solutionForallDescendants` returns *false*. Prior to that it weakens $b.U$ by introducing a disjunction with the returned formula U.
   If `optiSubtreeIsSolved`$(b, \mathcal{M})$ returns $\mathsf{UNSAT}(O)$, it means that $\mathcal{M} \not\models_{la} \mathcal{G}_b$. Since this is in agreement with the expected truth value ($\mathcal{M}(b.p) = \mathsf{false}$), we move on to the next descendant in $\mathsf{FAN}(n, \mathcal{M})$. Prior to that, `reasons` is strengthened by introducing a conjunction with $\neg b.p \Rightarrow \neg O$. If for no descendant nodes $b \in \mathsf{FAN}(n, \mathcal{M})$ the call `optiSubtreeIsSolved`$(b, \mathcal{M})$ returned $\mathsf{SAT}(U)$, function `solutionForallDescendants` returns *true*. Prior to that, formula `reasons` is strengthened by introducing a conjunction with $b.p$ for all descendant nodes $b \in \mathsf{NAN}(n, \mathcal{M})$.
   Two observations are in order. First, the strengthening of `reasons` replaces the strengthening of the $n.O$ formulas in QSMA: the OptiQSMA algorithm does not keep the $n.O$ formulas and it strengthens `reasons` instead. Second, the asymmetry in the strengthening of `reasons` (conjoined with $\neg b.p \Rightarrow \neg O$ for $b \in \mathsf{FAN}(n, \mathcal{M})$ and with $b.p$ for $b \in \mathsf{NAN}(n, \mathcal{M})$) reflects the asymmetry in the recursion: we recurse on first-alternation nodes and not on no-alternation nodes.
5. If `solutionForallDescendants` returns *true*, function `optiSubtreeIsSolved` builds formula L' as $LF(n) \wedge$ `reasons` (line 9 in Algorithm 4). The proof of partial correctness of `optiSubtreeIsSolved` (second bullet in the induction step in the proof of Theorem 6) shows that every model of formula L' fulfills the requirements for satisfaction with look-ahead (Definition 11). In this sense, `reasons` is an explanation of why a model is found with look-ahead. Then `optiSubtreeIsSolved` returns $\mathsf{SAT}(U)$, where U is the outcome of the application of MBU to L' and $\mathcal{M}$ (line 10).
   If `solutionForallDescendants` returns *false*, the control in function `optiSubtreeIsSolved` loops back to line 3. Since `solutionForallDescendants` returned *false*, it means that it found a node b in $\mathsf{FAN}(n, \mathcal{M})$ for which `optiSubtreeIsSolved`$(b,\mathcal{M})$ returned $\mathsf{SAT}(U)$ and the formula $b.U$ was updated (line 16). Therefore the state has changed, variable L gets a new formula on line 3, and the subsequent call to SMA will not produce the same model.

In the experiments it turned out that storing over-approximations for all nodes is less efficient than accumulating them temporarily in formula `reasons`, using `reasons` to compute formula L', and then forget the over-approximations. This is why the over-approximation O encapsulated in an $\mathsf{UNSAT}(O)$ value returned by a recursive call to `optiSubtreeIsSolved` is used but not saved.

5.3 Correctness of the **OptiQSMA** Algorithm

We begin by proving the partial correctness of function `optiSubtreeIsSolved`.

Theorem 6 (Partial Correctness)**.** *The function* `optiSubtreeIsSolved` *is partially correct: if the preconditions hold and the function halts, then the postconditions hold.*

Proof. Consider a call `optiSubtreeIsSolved`$(n, \mathcal{M})$, where $\mathcal{M}$ is an extension of $\mathcal{M}_0$ to the variables in $Rigid(n)$. We assume that the preconditions hold and the call terminates, and we show that the postconditions hold (see Algorithm 4 for pre and postconditions). The proof is by structural induction on the tree T_n in the QSMA-tree $\mathcal{G}_n = (T_n, Rigid(n))$ rooted at node n.
Base case: n is a leaf labeled $(\bar{x}, F[\bar{z}, \bar{x}])$ and $n.\psi = \exists \bar{x}.F[\bar{z}, \bar{x}]$. Formula L is assigned $LF(n) = n.F = F[\bar{z}, \bar{x}]$ since n has no descendants, and SMA is invoked to find an extension $\mathcal{M}'$ of $\mathcal{M}$ to $(FV(n.F) \setminus Rigid(n)) = \bar{x}$ such that $\mathcal{M}' \models n.F$. If no such extension exists (SMA returned *nil*), the call $\mathsf{MBO}(n.F, FV(n.F) \setminus Rigid(n), \mathcal{M})$ returns a quantifier-free formula O and `optiSubtreeIsSolved` returns $\mathsf{UNSAT}(O)$ (line 6 of Algorithm 4). By the specification of MBO, we have $\mathcal{M} \not\models O$ and $[\![n.\psi]\!] \subseteq [\![O]\!]$, so that the postconditions hold. If SMA returns an extension $\mathcal{M}'$ of $\mathcal{M}$ to $(FV(n.F) \setminus Rigid(n)) = \bar{x}$ such that $\mathcal{M}' \models n.F$, formula `reasons` is assigned $\top$. Since n has no descendants, `solutionForallDescendants` returns *true* leaving `reasons` unchanged. Thus, L' is assigned $LF(n) = n.F$, the call $\mathsf{MBU}(n.F, FV(n.F) \setminus Rigid(n), \mathcal{M})$ returns a quantifier-free formula U, and `optiSubtreeIsSolved` returns $\mathsf{SAT}(U)$ (line 10 of Algorithm 4). By the specification of MBU, we have $\mathcal{M} \models U$ and $[\![U]\!] \subseteq [\![n.\psi]\!]$, so that the postconditions hold.
Induction hypothesis: for all descendants b of node n, if the preconditions are satisfied and `optiSubtreeIsSolved`$(b, \mathcal{M})$ halts, the postconditions are satisfied.
Induction step: By induction hypothesis, for all descendants b of n, formula $b.U$ is an under-approximation of $b.\psi$ (i.e., the invariant of Algorithm 4 holds). Indeed, when $b.U$ is updated on line 16 of Algorithm 4, it gets $b.U \vee U$, where U is an under-approximation of $b.\psi$ returned by a recursive call `optiSubtreeIsSolved`$(b, \mathcal{M}')$. We distinguish two cases for the two exit points of `optiSubtreeIsSolved` (see Algorithm 4).

- Suppose `optiSubtreeIsSolved`$(n, \mathcal{M})$ returns $\mathsf{UNSAT}(O)$ on line 6, because SMA could not extend $\mathcal{M}$ to a model of

$$L = LF(n) \wedge \bigwedge_{n \to^+ b} (\neg b.p \Rightarrow \neg b.U).$$

  We must show that $[\![n.\psi]\!] \subseteq [\![O]\!]$ and $\mathcal{M} \not\models O$. The latter is directly a consequence of O being generated by MBO from L and $\mathcal{M}$. For the former, let $\mathcal{M}_O$ be a model of $n.\psi$ (i.e., $\mathcal{M}_O \models n.\psi$ and $\mathcal{M}_O \in [\![n.\psi]\!]$). It follows that $\mathcal{M}_O \models \mathcal{G}_n$ by Theorem 1 and $\mathcal{M}_O \models_{la} \mathcal{G}_n$ by Theorem 5. By Definition 11, $\mathcal{M}_O$ can be extended into a model $\mathcal{M}'_O$ of $LF(n)$ such that for all $b \in \mathsf{FAN}(n, \mathcal{M}'_O)$, $\mathcal{M}'_O \not\models_{la} \mathcal{G}_b$. It follows that $\mathcal{M}'_O \not\models b.\psi$ by Theorem 1, and hence $\mathcal{M}'_O \not\models b.U$ by the invariant in the precondition, so that $\mathcal{M}'_O \models \neg b.U$ (*).
  Now we have that $\mathcal{M}'_O \models LF(n)$ and we want to show that $\mathcal{M}'_O \models L$, in order to get $\mathcal{M}'_O \models O$ as O is an over-approximation of L generated by MBO. Towards

showing that $\mathcal{M}'_O \models L$, we assume that $\mathcal{M}'_O(c.p) = \mathsf{true}$ for all descendants c of n beyond the first alternation nodes, that is, for all nodes c such that $n \to^+ c$ and $c \notin \mathsf{NAN}(n, \mathcal{M}'_O) \cup \mathsf{FAN}(n, \mathcal{M}'_O)$ (†).

We show that this assumption causes no loss of generality. Indeed, forcing $\mathcal{M}'_O(c.p) = \mathsf{true}$ for such nodes affects neither $\mathsf{NAN}(n, \mathcal{M}'_O)$, nor $\mathsf{FAN}(n, \mathcal{M}'_O)$, nor $\mathcal{M}'_O(b.p) = \mathsf{false}$ for all $b \in \mathsf{FAN}(n, \mathcal{M}'_O)$. Also, this assumption does not affect the fact that $\mathcal{M}'_O \models LF(n)$. Indeed, $LF(n)$ has the form:

$$n.F \ \wedge \bigwedge_{n \to^+ b} \{n_1.p \Rightarrow (n_1.F \ \wedge \ n_2.p \Rightarrow (n_2.F \ \wedge \ n_3.p \Rightarrow (\cdots \\ n_{q-1}.F \ \wedge \ n_q.p \Rightarrow (n_q.F \ \wedge \ b.p \Rightarrow b.F) \cdots))) \mid n \to n_1 \to \cdots \to n_q \to b\}.$$

Therefore, forcing $\mathcal{M}'_O(c.p) = \mathsf{true}$ for every node c that is below some node $b \in \mathsf{FAN}(n, \mathcal{M}'_O)$ does not affect the truth value of $LF(n)$, because $\mathcal{M}'_O(b.p) = \mathsf{false}$ and hence any implication in $LF(n)$ involving a $c.p$ evaluates to true regardless of the truth value of $c.p$.

Next, we show that indeed $\mathcal{M}'_O \models L$. First, we already have $\mathcal{M}'_O \models LF(n)$. Second, we have also $\mathcal{M}'_O \models (\neg b.p \Rightarrow \neg b.U)$ for all descendants b of n: if $b \in \mathsf{FAN}(n, \mathcal{M}'_O)$ then $\mathcal{M}'_O(b.p) = \mathsf{false}$ and we know that $\mathcal{M}'_O \models \neg b.U$ by (*); if $b \in \mathsf{NAN}(n, \mathcal{M}'_O)$ then $\mathcal{M}'_O(b.p) = \mathsf{true}$, so that $\mathcal{M}'_O(\neg b.p \Rightarrow \neg b.U) = \mathsf{true}$; and if $b \notin \mathsf{NAN}(n, \mathcal{M}'_O) \cup \mathsf{FAN}(n, \mathcal{M}'_O)$, then $\mathcal{M}'_O(b.p) = \mathsf{true}$ by the assumption (†), so that $\mathcal{M}'_O(\neg b.p \Rightarrow \neg b.U) = \mathsf{true}$.

From $\mathcal{M}'_O \models L$ we get $\mathcal{M}_O \models L$. Then, since O is generated by MBO from L and $\mathcal{M}$, by the specification of MBO we know that L implies O in the theory. Thus, from $\mathcal{M}_O \models L$, it follows that $\mathcal{M}_O \models O$. Therefore, also the postcondition $[\![n.\psi]\!] \subseteq [\![O]\!]$ holds.

- Suppose `optiSubtreeIsSolved`$(n, \mathcal{M})$ returns $\mathsf{SAT}(U)$ on line 10 of Algorithm 4. Function SMA found an extension $\mathcal{M}'$ satisfying L, and hence $LF(n)$. Furthermore, `solutionForallDescendants` returned *true* after constructing a formula

$$\texttt{reasons} \quad = \quad (\bigwedge_{b \in \mathsf{NAN}(n,\mathcal{M}')} b.p) \wedge (\bigwedge_{b \in \mathsf{FAN}(n,\mathcal{M}')} (\neg b.p \Rightarrow \neg O_b))$$

where, for all $b \in \mathsf{FAN}(n, \mathcal{M}')$, formula O_b is an over-approximation of $b.\psi$ that was returned as $\mathsf{UNSAT}(O_b)$ by a recursive call `optiSubtreeIsSolved`$(b, \mathcal{M}')$. By the post-condition of that recursive call, $\mathcal{M}' \not\models O_b$. By Theorem 1, $\mathcal{M}' \not\models \mathcal{G}_b$. Since this holds for all $b \in \mathsf{FAN}(n, \mathcal{M}')$, we have that $\mathcal{M}'$ fulfills Definition 11.

As promised (see Step (5) in the description of `optiSubtreeIsSolved`), we show that this property holds in general: every model that satisfies $L' = (LF(n) \wedge \texttt{reasons})$ fulfills Definition 11. It suffices to show that every model that satisfies `reasons` fulfills Condition (ii) in Definition 11. Let $\mathcal{M}''$ be a model that satisfies `reasons`. It follows that $\mathsf{NAN}(n, \mathcal{M}'') = \mathsf{NAN}(n, \mathcal{M}')$, $\mathsf{FAN}(n, \mathcal{M}'') = \mathsf{FAN}(n, \mathcal{M}')$, and for all $b \in \mathsf{FAN}(n, \mathcal{M}')$, $\mathcal{M}'' \models \neg O_b$. It follows that for all $b \in \mathsf{FAN}(n, \mathcal{M}')$, $\mathcal{M}'' \not\models O_b$ and hence by induction hypothesis $\mathcal{M}'' \not\models b.\psi$. By Theorem 1, we have that for all $b \in \mathsf{FAN}(n, \mathcal{M}')$, $\mathcal{M}'' \not\models \mathcal{G}_b$. Thus, $\mathcal{M}'$ fulfills Condition (ii) of Definition 11.

By the specification of MBU, the application of MBU to L' and $\mathcal{M}$ yields a quantifier-free formula U such that $\mathcal{M} \models U$ and U implies L' in the theory. Therefore, for all $\mathcal{M}_U \in [\![U]\!]$, model $\mathcal{M}_U$ can be extended into a model that satisfies L', and hence fulfills Definition 11 by the argument above. This means that for all models $\mathcal{M}_U \in [\![U]\!]$, $\mathcal{M}_U \models_{la} \mathcal{G}_n$, so that $\mathcal{M}_U \models \mathcal{G}_n$ by Theorem 5, and $\mathcal{M}_U \models n.\psi$ by Theorem 1. This shows that $[\![n.U]\!] \subseteq [\![n.\psi]\!]$, so that the postconditions hold.

□

We prove next the termination of function `optiSubtreeIsSolved`, thus establishing the total correctness of the OptiQSMA algorithm.

Theorem 7 (Termination)**.** *If the* MBU *and* MBO *functions have finite basis, whenever the preconditions are satisfied the function* `optiSubtreeIsSolved` *halts.*

Proof. Consider a call `optiSubtreeIsSolved`$(n, \mathcal{M})$, where $\mathcal{M}$ is an extension of $\mathcal{M}_0$ to $\mathit{Rigid}(n)$. We assume that the preconditions hold (see Algorithm 4) and we show that the call terminates. Let $\mathcal{G}_n = (T_n, \mathit{Rigid}(n))$ be the QSMA-tree rooted at node n. For all nodes b in T_n, including n itself, using $\bar{z}_b$ for $\mathit{Rigid}(b)$, we build finite sets

- $\mathcal{U}_b = \{U^b_1[\bar{z}_b], \ldots, U^b_{l_b}[\bar{z}_b]\}$ of under-approximations of formula $b.\psi$, and
- $\mathcal{O}_b = \{O^b_1[\bar{z}_b], \ldots, O^b_{m_b}[\bar{z}_b]\}$ of over-approximations of formula $b.\psi$,

such that

1. For all nodes b in T_n, including n itself, formula $b.U$ is a disjunction of the elements in a subset of $\mathcal{U}_b$, and
2. The call `optiSubtreeIsSolved`$(n, \mathcal{M})$ halts returning either $\mathsf{SAT}(U)$ for some $U \in \mathcal{U}_n$ or $\mathsf{UNSAT}(O)$ for some $O \in \mathcal{O}_n$.

The proof of Claims (1) and (2) and the construction of the sets of under- and over-approximations are by structural induction on the tree T_n.
Base case: n is a leaf labeled $(\bar{x}, F[\bar{z}, \bar{x}])$, where $\bar{z} = \mathit{Rigid}(n)$, and $n.\psi = \exists\bar{x}.F[\bar{z}, \bar{x}]$. Let $\mathcal{U}_n$ and $\mathcal{O}_n$ be the finite bases of functions MBU and MBO, respectively, for formula $n.F = F[\bar{z}, \bar{x}]$ and variable tuple $(FV(n.F) \setminus \mathit{Rigid}(n)) = \bar{x}$. In symbols, let

$$\mathcal{U}_n = \{\mathsf{MBU}(F[\bar{z}, \bar{x}], \bar{x}, \mathcal{M}) \mid \mathcal{M} : \text{extension of } \mathcal{M}_0 \text{ to } \bar{z} \text{ such that } \mathcal{M} \models \exists\bar{x}.F[\bar{z}, \bar{x}]\}$$

and

$$\mathcal{O}_n = \{\mathsf{MBO}(F[\bar{z}, \bar{x}], \bar{x}, \mathcal{M}) \mid \mathcal{M} : \text{extension of } \mathcal{M}_0 \text{ to } \bar{z} \text{ such that } \mathcal{M} \not\models \exists\bar{x}.F[\bar{z}, \bar{x}]\}.$$

By the finite basis hypothesis, $\mathcal{U}_n$ and $\mathcal{O}_n$ are finite, and they are sets of under- and over-approximations of formula $n.\psi$ as required. Since n is a leaf, it is the only node in T_n, and $n.U = \bot$, which is the disjunction of the elements in $\emptyset \subseteq \mathcal{U}_n$, so that Claim (1) holds. For Claim (2), since n is a leaf, there is no recursion, and termination is trivial. Furthermore, `optiSubtreeIsSolved` returns either $\mathsf{SAT}(U)$ for some $U \in \mathcal{U}_n$ or $\mathsf{UNSAT}(O)$ for some $O \in \mathcal{O}_n$. Indeed, formula L is assigned $LF(n) = n.F = F[\bar{z}, \bar{x}]$, and SMA is invoked to find an extension $\mathcal{M}'$ of $\mathcal{M}$ to $(FV(n.F) \setminus \mathit{Rigid}(n)) = \bar{x}$ such that $\mathcal{M}' \models n.F$. If SMA returns *nil*, the call $\mathsf{MBO}(n.F, FV(n.F) \setminus \mathit{Rigid}(n), \mathcal{M})$ returns a quantifier-free formula $O \in \mathcal{O}_n$ and `optiSubtreeIsSolved` returns $\mathsf{UNSAT}(O)$. If SMA returns an extension $\mathcal{M}'$ of $\mathcal{M}$

to $(FV(n.F) \setminus Rigid(n)) = \bar{x}$ such that $\mathcal{M}' \models n.F$, formula `reasons` is assigned $\top$, `solutionForallDescendants` returns *true* leaving `reasons` unchanged, L' is assigned $LF(n) = n.F$, the call $\mathsf{MBU}(n.F, FV(n.F) \setminus Rigid(n), \mathcal{M})$ returns a quantifier-free formula $U \in \mathcal{U}_n$, and `optiSubtreeIsSolved` returns $\mathsf{SAT}(U)$.
Induction hypothesis: for all descendants b of node n, the claims hold; that is, if MBU and MBO have finite basis and the preconditions are satisfied, the finite sets $\mathcal{U}_b$ and $\mathcal{O}_b$ have been constructed, and Claims (1) and (2) hold.
Induction step: we prove Claims (1) and (2) and we construct $\mathcal{U}_n$ and $\mathcal{O}_n$ simultaneously. We prove Claim (1) by showing that it is an invariant of function `optiSubtreeIsSolved`: we assume that Claim (1) holds as a precondition, and we show that it remains true throughout the execution of both `optiSubtreeIsSolved` and `solutionForallDescendants`. To this end, we show that Claim (1) is an invariant for the `while` loop in `optiSubtreeIsSolved` and the `for` loop ranging over all $b \in \mathsf{FAN}(n, \mathcal{M})$ in `solutionForallDescendants`: we assume that Claim (1) holds at the beginning of either loop body and we show that it is preserved.
We prove Claim (2) by showing that both functions `optiSubtreeIsSolved` and `solutionForallDescendants` halt and `optiSubtreeIsSolved`$(n, \mathcal{M})$ returns either $\mathsf{SAT}(U)$ for some $U \in \mathcal{U}_n$ or $\mathsf{UNSAT}(O)$ for some $O \in \mathcal{O}_n$.
Consider a call `optiSubtreeIsSolved`$(b, \mathcal{M})$ in the `for` loop. Assume that the preconditions of `optiSubtreeIsSolved`$(b, \mathcal{M})$ are satisfied. Then by induction hypothesis `optiSubtreeIsSolved`$(b, \mathcal{M})$ terminates, returning either $\mathsf{SAT}(U)$ for some $U \in \mathcal{U}_b$ or $\mathsf{UNSAT}(O)$ for some $O \in \mathcal{O}_b$. Whenever $b.U$ is updated by instruction $b.U \leftarrow b.U \vee U$ (line 16 of Algorithm 4), formula $b.U$ remains a disjunction of elements in a subset of $\mathcal{U}_b$, so that Claim (1) is preserved. Therefore, Claim (1) is an invariant of `solutionForallDescendants` and `solutionForallDescendants` terminates. Since no instruction in the body of the `while` loop modifies $b.U$ for any node b, Claim (1) is an invariant also for the `while` loop.
We prove next the termination of the `while` loop. Invariant (1) implies that for all descendants b of n, formula $b.U$ can be updated at most $|\mathcal{U}_b|$ times, where $|\mathcal{U}_b|$ is the cardinality of the set $\mathcal{U}_b$. By way of contradiction, suppose that the `while` loop does not halt. This means that it generates an infinite series of calls to `solutionForallDescendants` each returning *false*. Since each call to `solutionForallDescendants` that returns *false* updates some $b.U$ at least once, such an infinite series contradicts the fact that for each descendant node b only finitely many updates to $b.U$ are available.
To complete the proof, we observe that for all nodes b in T_n, the space of possible values for $b.U$ is finite, because $b.U$ is a disjunction of the elements in a subset of $\mathcal{U}_b$. If $|\mathcal{U}_b| = v_b$, the number of possible values for $b.U$ is given by the number of i-combinations out of an alphabet of v_b elements:

$$\sum_{i=0}^{v_b} \binom{v_b}{i} = \sum_{i=0}^{v_b} \frac{v_b!}{i! \cdot (v_b - i)!}.$$

This implies that also the space of possible values for L is finite (see line 3 in Algorithm 4). Therefore, MBO gets applied to finitely many formulas and variable

tuples. By the finite basis hypothesis, for each (formula, tuple) pair, MBO can produce only finitely many formulas that we all add to $\mathcal{O}_n$. By construction, whenever `optiSubtreeIsSolved`$(n, \mathcal{M})$ returns $\mathsf{UNSAT}(O)$, formula O is in $\mathcal{O}_n$.
By the finiteness of $\mathcal{O}_n$, also the space of possible values for the variable `reasons` is finite (see line 19 in Algorithm 4). This implies that also the space of possible values for L' is finite (see line 9 in Algorithm 4). Therefore, MBU gets applied to finitely many formulas and variable tuples. By the finite basis hypothesis, for each (formula, tuple) pair, MBU can produce only finitely many formulas that we all add to $\mathcal{U}_n$. By construction, whenever `optiSubtreeIsSolved`$(n, \mathcal{M})$ returns $\mathsf{SAT}(U)$, formula U is in $\mathcal{U}_n$, which concludes the proof. □

6 MBU and MBO Functions for LRA

In this section we consider the theory LRA, so that $\mathcal{M}_0$ is the unique model of LRA, where sort Q is interpreted as the set $\mathbb{Q}$ of the rational numbers, and all constant, function, and predicate symbols in the LRA signature are interpreted as usual. We exhibit MBU and MBO functions for LRA, named $\mathsf{MBU}_{\mathsf{LRA}}$ and $\mathsf{MBO}_{\mathsf{LRA}}$. Both functions rely on the *Fourier-Motzkin (FM) resolution* rule [29, 30]:

$$t_1 \lhd_1 x,\ x \blacktriangleleft t_2 \vdash_{\mathsf{LRA}} t_1 \lhd_2 t_2,$$

where the variable x and the terms t_1, t_2, and t_3 have sort Q, and $\lhd_1, \blacktriangleleft, \lhd_2 \in \{<, \leq\}$, with the proviso that $\lhd_2$ is $<$ if either $\lhd_1$ or $\blacktriangleleft$ is $<$ and it is $\leq$ otherwise. Here we use $\lhd$ for lower bounds on x, and for inequalities that do not involve x, and $\blacktriangleleft$ for upper bounds on x, because it will be convenient for the description of $\mathsf{MBU}_{\mathsf{LRA}}$ and $\mathsf{MBO}_{\mathsf{LRA}}$. A systematic application of FM-resolution to decide LRA-satisfiability is doubly exponential in the worst case (e.g., [29]). FM-resolution is still relevant if applied sparingly only to explain conflicts in conflict-driven satisfiability procedures (see [8, Section 4.4] for a discussion and references). Here we consider FM-resolution only for the definition of MBU and MBO functions for LRA. Also for this end FM-resolution may be too inefficient, because it generates too many atoms. The $\mathsf{MBU}_{\mathsf{LRA}}$ and $\mathsf{MBO}_{\mathsf{LRA}}$ functions presented here have an illustrative purpose.

6.1 A Model-Based Under-Approximation Function for LRA

An MBU function for a theory takes as arguments a quantifier-free formula $F[\bar{z}, x]$, a tuple of variables $\bar{x}$, and an extension $\mathcal{M}$ of $\mathcal{M}_0$ to $\bar{z}$ such that $\mathcal{M} \models \exists x.F[\bar{z}, x]$. The definition of an MBU function only requires to consider the case where the formula $F[\bar{z}, x]$ is a conjunction of literals and the tuple $\bar{x}$ is a single variable x. The other cases can be reduced to this one by a set of theory-independent reduction rules [3]. Let a denote an atom and $sign(a, \mathcal{M})$ be defined as follows:

$$sign(a, \mathcal{M}) = \begin{cases} a & \text{if } \mathcal{M} \models a, \\ \neg a & \text{otherwise.} \end{cases}$$

Let G and H denote conjunctions of literals. Then the reduction rules are:

$$\begin{aligned}
\mathsf{MBU}(F[\bar{z},\bar{x}x],\bar{x}x,\mathcal{M}) &= \mathsf{MBU}(\mathsf{MBU}(F[\bar{z},\bar{x}x],x,\mathcal{M}),\bar{x},\mathcal{M})\\
\mathsf{MBU}(F[\bar{z}],\emptyset,\mathcal{M}) &= F[\bar{z}]\\
\mathsf{MBU}(F[\bar{z},\bar{x}],\bar{x},\mathcal{M}) &= \mathsf{MBU}(\bigwedge\{sign(a,\mathcal{M}) \mid a \text{ occurs in } F[\bar{z},\bar{x}]\},\bar{x},\mathcal{M})\\
\mathsf{MBU}(G[\bar{z},x]\wedge H[\bar{z}],x,\mathcal{M}) &= \mathsf{MBU}(G[\bar{z},x],x,\mathcal{M})\wedge H[\bar{z}].
\end{aligned}$$

The first two rules reduce working with a tuple of variables $\bar{x}$ to working with one variable x at a time. The third rule reduces the computation of MBU for model $\mathcal{M}$ and arbitrary quantifier-free formula $F[\bar{z},\bar{x}]$ to the computation of MBU for $\mathcal{M}$ and the conjunction of the literals whose atom occurs in $F[\bar{z},\bar{x}]$ and whose sign is dictated by $\mathcal{M}$. Such a conjunction is true in $\mathcal{M}$ and it is stronger than $F[\bar{z},\bar{x}]$, so that an under-approximation of the conjunction is an under-approximation of $F[\bar{z},\bar{x}]$. The fourth rule says that once working on a conjunction of literals and a single variable, it is possible to skip the literals where the variable does not occur. If MBU has finite basis for one variable x and conjunction $l_1\wedge\ldots\wedge l_n$ of literals where x occurs in each literal, then MBU has finite basis in all cases.

We define the $\mathsf{MBU}_{\mathsf{LRA}}$ function, assuming that the tuple $\bar{x}$ is a single variable x and the formula $F[\bar{z},x]$ is a conjunction of literals where x occurs in each literal. The predicate symbols in the signature of LRA are equality $\simeq$, inequality $<$, and weak inequality $\leq$. Given $F[\bar{z},x]=l_1\wedge\ldots\wedge l_n$, first the literals are rewritten so that each of them is of the form $x\simeq t$, $x\not\simeq t$, $x<t$, $x\leq t$, $t<x$, or $t\leq x$, for t a term of sort Q such that $x\notin FV(t)$. Then, $\mathsf{MBU}_{\mathsf{LRA}}(F[\bar{z},x],x,\mathcal{M})$ is generated as follows:

1. If $F[\bar{z},x]$ contains a literal l_i of the form $x\simeq t$, then

   $$\mathsf{MBU}_{\mathsf{LRA}}(F[\bar{z},x],x,\mathcal{M})=(l_1\wedge\ldots\wedge l_{i-1}\wedge l_{i+1}\wedge\ldots\wedge l_n)\{x\leftarrow t\},$$

   where $\{x\leftarrow t\}$ is the substitution that replaces x with t. In words, the literal $x\simeq t$ is eliminated and x is replaced by t everywhere.
2. Otherwise, for all literals l_i of the form $x\not\simeq t$, literal l_i is replaced by $x<t$, if $\mathcal{M}(x)<\mathcal{M}(t)$, and by $t<x$, if $\mathcal{M}(t)<\mathcal{M}(x)$;
3. Then, the literals in $F[\bar{z},x]$ are reordered as follows:
   (a) All lower bounds on x are listed before all upper bounds on x:
   $F[\bar{z},x]=t_1\lhd_1 x\wedge\ldots\wedge t_n\lhd_n x\wedge x\blacktriangleleft_1 s_1\wedge\ldots\wedge x\blacktriangleleft_m s_m$
   where $\forall i,\ 1\leq i\leq n,\ \lhd_i\in\{<,\leq\}$ and $\forall j,\ 1\leq j\leq m,\ \blacktriangleleft_j\in\{<,\leq\}$;
   (b) The lower bound literals are ordered in non-increasing order of the evaluation of the bounds in $\mathcal{M}$, and the upper bound literals are ordered in non-decreasing order of the evaluation of the bounds in $\mathcal{M}$:
   $F[\bar{z},x]=t_1\lhd_1 x\wedge\ldots\wedge t_n\lhd_n x\wedge x\blacktriangleleft_1 s_1\wedge\ldots\wedge x\blacktriangleleft_m s_m$
   where
   - $\forall i,\ 1\leq i\leq n,\ \mathcal{M}(t_i)\geq\mathcal{M}(t_{i+1})$, and if $\mathcal{M}(t_i)=\mathcal{M}(t_{i+1})$ and $\lhd_i\neq\lhd_{i+1}$ then $\lhd_i$ is $<$ and $\lhd_{i+1}$ is $\leq$; and
   - $\forall j,\ 1\leq j\leq m,\ \mathcal{M}(s_j)\leq\mathcal{M}(s_{j+1})$, and if $\mathcal{M}(s_j)=\mathcal{M}(s_{j+1})$ and $\blacktriangleleft_j\neq\blacktriangleleft_{j+1}$ then $\blacktriangleleft_j$ is $<$ and $\blacktriangleleft_{j+1}$ is $\leq$.

4. At this stage, $\mathsf{MBU}_{\mathsf{LRA}}(F[\bar{z},x],x,\mathcal{M})$ is defined as follows:

$$\mathsf{MBU}_{\mathsf{LRA}}(F[\bar{z},x],x,\mathcal{M}) = \begin{cases} \bigwedge_{i=2}^{n} t_i \leq t_1 \wedge \bigwedge_{j=2}^{m} s_1 \leq s_j \wedge t_1 \lhd s_1 & \text{if } n>0,\ m>0,\\ \bigwedge_{i=2}^{n} t_i \leq t_1 & \text{if } m=0,\\ \bigwedge_{j=2}^{m} s_1 \leq s_i & \text{if } n=0, \end{cases}$$

where $\lhd$ is $<$ iff either $\lhd_1$ or $\blacktriangleleft_1$ is $<$.

The following theorem shows that $\mathsf{MBU}_{\mathsf{LRA}}$ satisfies the desiderata.

Theorem 8. *For all* LRA*-formulas* $\exists x.F[\bar{z},x]$*, where* $F[\bar{z},x]$ *is a quantifier-free conjunction of literals where* x *occurs in each literal, and all extensions* $\mathcal{M}$ *of* $\mathcal{M}_0$ *to* $\bar{z}$ *such that* $\mathcal{M} \models \exists x.F[\bar{z},x]$*, the call* $\mathsf{MBU}_{\mathsf{LRA}}(F[\bar{z},x],x,\mathcal{M})$ *returns a quantifier-free formula* $U_{\mathsf{LRA}}[\bar{z}]$ *such that* $\mathcal{M} \models U_{\mathsf{LRA}}[\bar{z}]$ *and* $\mathsf{LRA} \models U_{\mathsf{LRA}}[\bar{z}] \Rightarrow (\exists x.F[\bar{z},x])$*. Furthermore, the* $\mathsf{MBU}_{\mathsf{LRA}}$ *function has finite basis.*

Proof. For the first claim, we distinguish two cases, depending on whether $U_{\mathsf{LRA}}[\bar{z}]$ is produced at Step (1) or Step (4) of the above construction:

- $U_{\mathsf{LRA}}[\bar{z}]$ is the conjunction produced at Step (1):
  We show that $\mathcal{M} \models U_{\mathsf{LRA}}[\bar{z}]$. By hypothesis, $\mathcal{M} \models \exists x.F[\bar{z},x]$. Thus, there exists a value $q \in \mathbb{Q}$ for x such that $\mathcal{M} \models F[\bar{z},x]$ with $\mathcal{M}(x) = q$. Since the literal $x \simeq t$ (where x does not occur in t) is in the conjunction $F[\bar{z},x]$, this means that $\mathcal{M}(t) = q$. For all literals l other than $x \simeq t$ in $F[\bar{z},x]$, $\mathcal{M} \models l$ with $\mathcal{M}(x) = q = \mathcal{M}(t)$. Therefore, for all such literals l we have $\mathcal{M} \models l\{x \leftarrow t\}$, and hence $\mathcal{M} \models U_{\mathsf{LRA}}[\bar{z}]$.
  We show that $\mathsf{LRA} \models U_{\mathsf{LRA}}[\bar{z}] \Rightarrow (\exists x.F[\bar{z},x])$. We need to show that for all extensions $\mathcal{M}'$ of $\mathcal{M}_0$ to $\bar{z}$ such that $\mathcal{M}' \models U_{\mathsf{LRA}}[\bar{z}]$, it is also the case that $\mathcal{M}' \models \exists x.F[\bar{z},x]$. Assume that $\mathcal{M}' \models U_{\mathsf{LRA}}[\bar{z}]$. Since $F[\bar{z},x]$ differs from $U_{\mathsf{LRA}}[\bar{z}]$ only by the addition of the literal $x \simeq t$ and the abstraction that replaces occurrences of term t with variable x, it suffices to interpret x as $\mathcal{M}'(t)$ to have that $\mathcal{M}' \models \exists x.F[\bar{z},x]$.
- $U_{\mathsf{LRA}}[\bar{z}]$ is the conjunction produced at Step (4):
  We show that $\mathcal{M} \models U_{\mathsf{LRA}}[\bar{z}]$. By hypothesis, $\mathcal{M} \models \exists x.F[\bar{z},x]$. Step (2) preserves the truth in $\mathcal{M}$ of the conjunction of literals. By the reordering in Step (3), the lower bound $t_1 \lhd_1 x$ implies in $\mathcal{M}$ the lower bounds $t_i \lhd_i x$ for $i = 2,\ldots,n$, or, equivalently, the conjunction $\bigwedge_{i=2}^{n} t_i \leq t_1$ in $U_{\mathsf{LRA}}[\bar{z}]$ is true in $\mathcal{M}$. Similarly, the upper bound $x \blacktriangleleft_1 s_1$ implies in $\mathcal{M}$ the upper bounds $x \blacktriangleleft_j s_j$ for $j = 2,\ldots,m$, or, equivalently, the conjunction $\bigwedge_{j=2}^{m} s_1 \leq s_j$ in $U_{\mathsf{LRA}}[\bar{z}]$ is true in $\mathcal{M}$. Last, $t_1 \lhd s_1$ follows from $t_1 \lhd_1 x$ and $x \blacktriangleleft_1 s_1$ by an FM-resolution step, which is sound in LRA, so that the literal $t_1 \lhd s_1$ in $U_{\mathsf{LRA}}[\bar{z}]$ is also true in $\mathcal{M}$.
  We show that $\mathsf{LRA} \models U_{\mathsf{LRA}}[\bar{z}] \Rightarrow (\exists x.F[\bar{z},x])$. We need to show that for all extensions $\mathcal{M}'$ of $\mathcal{M}_0$ to $\bar{z}$ such that $\mathcal{M}' \models U_{\mathsf{LRA}}[\bar{z}]$, it is also the case that $\mathcal{M}' \models \exists x.F[\bar{z},x]$. Assume that $\mathcal{M}' \models U_{\mathsf{LRA}}[\bar{z}]$. The truth of $\bigwedge_{i=2}^{n} t_i \leq t_1$ in $\mathcal{M}'$ means that finding a value $q \in \mathbb{Q}$ for x satisfying $t_1 \lhd_1 x$ suffices to satisfy all the lower bounds on x in $F[\bar{z},x]$. This suffices for $\mathcal{M}' \models \exists x.F[\bar{z},x]$ if $m = 0$, because in this case $F[\bar{z},x]$ contains only lower bounds on x, and a value for x satisfying them all certainly exists in $\mathbb{Q}$. The truth of $\bigwedge_{j=2}^{m} s_1 \leq s_j$ in $\mathcal{M}'$ means that finding a value $q \in \mathbb{Q}$ for x satisfying $x \blacktriangleleft_1 s_1$ suffices to satisfy all the upper bounds on x in $F[\bar{z},x]$. This

suffices for $\mathcal{M}' \models \exists x.F[\bar{z}, x]$ if $n = 0$, because in this case $F[\bar{z}, x]$ contains only upper bounds on x, and a value for x satisfying them all certainly exists in $\mathbb{Q}$. If $n > 0$ and $m > 0$, the truth of $t_1 \lhd s_1$ in $\mathcal{M}'$ means that a value $q \in \mathbb{Q}$ for x satisfying $t_1 \lhd_1 x$ and $x \blacktriangleleft_1 s_1$ does exist. Therefore, $\mathcal{M}' \models \exists x.F[\bar{z}, x]$.

For the second claim, the $\mathsf{MBU}_{\mathsf{LRA}}$ function has finite basis, because for given $F[\bar{z}, x]$ and x, as the extension $\mathcal{M}$ varies, we get different evaluations of the bounds on x in $F[\bar{z}, x]$, and hence different permutations of the literals in $F[\bar{z}, x]$. Since there are finitely many literals in $F[\bar{z}, x]$, the number of their permutations is finite. Thus, $\mathsf{MBU}_{\mathsf{LRA}}$ produces finitely many formulas for given $F[\bar{z}, x]$ and x. □

6.2 A Model-Based Over-Approximation Function for LRA

An MBO function for a theory takes as arguments a quantifier-free formula $F[\bar{z}, x]$, a tuple of variables $\bar{x}$, and an extension $\mathcal{M}$ of $\mathcal{M}_0$ to $\bar{z}$ such that $\mathcal{M} \not\models \exists x.F[\bar{z}, x]$. Similar to the MBU case, also for the definition of an MBO for a theory, it suffices to consider the case where the formula $F[\bar{z}, x]$ is a conjunction of literals, and the tuple $\bar{x}$ is a single variable x. The reason is that MBO computation distributes over disjunction. Indeed, first $\exists \bar{x}.(F_1[\bar{z}, \bar{x}] \vee F_2[\bar{z}, \bar{x}])$ is equivalent to $\exists \bar{x}.F_1[\bar{z}, \bar{x}] \vee \exists \bar{x}.F_2[\bar{z}, \bar{x}]$. Second, if $O_1[\bar{z}]$ is an MBO of $\exists \bar{x}.F_1[\bar{z}, \bar{x}]$ with respect to extension $\mathcal{M}$, and $O_2[\bar{z}]$ is an MBO of $\exists \bar{x}.F_2[\bar{z}, \bar{x}]$ with respect to $\mathcal{M}$, then $O_1[\bar{z}] \vee O_2[\bar{z}]$ is an MBO of $\exists \bar{x}.F_1[\bar{z}, \bar{x}] \vee \exists \bar{x}.F_2[\bar{z}, \bar{x}]$ with respect to $\mathcal{M}$, because $\mathcal{M} \not\models O_1[\bar{z}]$ and $\mathcal{M} \not\models O_2[\bar{z}]$ imply $\mathcal{M} \not\models O_1[\bar{z}] \vee O_2[\bar{z}]$, and $\mathcal{T} \models (\exists \bar{x}.F_1[\bar{z}, \bar{x}]) \Rightarrow O_1[\bar{z}]$ and $\mathcal{T} \models (\exists \bar{x}.F_2[\bar{z}, \bar{x}]) \Rightarrow O_2[\bar{z}]$ imply $\mathcal{T} \models (\exists \bar{x}.F_1[\bar{z}, \bar{x}] \vee \exists \bar{x}.F_2[\bar{z}, \bar{x}]) \Rightarrow (O_1[\bar{z}] \vee O_2[\bar{z}])$.

Therefore, an arbitrary quantifier-free formula $F[\bar{z}, x]$ is transformed into Disjunctive Normal Form, denoted $\mathit{DNF}(F)[\bar{z}, x]$, and the MBO is obtained by taking the disjunction of the MBO's of the disjuncts in $\mathit{DNF}(F)[\bar{z}, x]$, which are conjunctions of literals. Let G, H, and G_i, for all i, $1 \leq i \leq n$, be conjunctions of literals. The third and fourth rules in the following set achieve this reduction, while the other rules correspond to those given for MBU:

$$\begin{aligned}
\mathsf{MBO}(F[\bar{z}, \bar{x}x], \bar{x}x, \mathcal{M}) &= \mathsf{MBO}(\mathsf{MBO}(F[\bar{z}, \bar{x}x], x, \mathcal{M}), \bar{x}, \mathcal{M}) \\
\mathsf{MBO}(F[\bar{z}], \emptyset, \mathcal{M}) &= F[\bar{z}] \\
\mathsf{MBO}(F[\bar{z}, \bar{x}], \bar{x}, \mathcal{M}) &= \mathsf{MBO}(\mathit{DNF}(F)[\bar{z}, \bar{x}], \bar{x}, \mathcal{M}) \\
\mathsf{MBO}(\bigvee_{i=1}^{n} G_i[\bar{z}, \bar{x}], \bar{x}, \mathcal{M}) &= \bigvee_{i=1}^{n} \mathsf{MBO}(G_i[\bar{z}, \bar{x}], \bar{x}, \mathcal{M}) \\
\mathsf{MBO}(G[\bar{z}, x] \wedge H[\bar{z}], x, \mathcal{M}) &= \mathsf{MBO}(G[\bar{z}, x], x, \mathcal{M}) \wedge H[\bar{z}].
\end{aligned}$$

Similar to MBU, if MBO has finite basis for one variable x and conjunction $l_1 \wedge \ldots \wedge l_n$ of literals where x occurs in each literal, then MBO has finite basis in all cases.

We define the $\mathsf{MBO}_{\mathsf{LRA}}$ function, assuming that the tuple $\bar{x}$ is a single variable x and the formula $F[\bar{z}, x]$ is a conjunction of literals where x occurs in each literal. Given $F[\bar{z}, x] = l_1 \wedge \ldots \wedge l_n$, first the literals are rewritten so that each of them is of the form $x \simeq t$, $x \not\simeq t$, $x < t$, $x \leq t$, $t < x$, or $t \leq x$, for t a term of sort Q such that $x \notin FV(t)$. Then, $\mathsf{MBO}_{\mathsf{LRA}}(F[\bar{z}, x], x, \mathcal{M})$ is generated as follows:

1. If $F[\bar{z}, x]$ contains a literal l_i of the form $x \simeq t$, then

$$\mathsf{MBO}_{\mathsf{LRA}}(F[\bar{z}, x], x, \mathcal{M}) = (l_1 \wedge \ldots \wedge l_{i-1} \wedge l_{i+1} \wedge \ldots \wedge l_n)\{x \leftarrow t\},$$

   where $\{x \leftarrow t\}$ is the substitution that replaces x with t. In words, the literal $x \simeq t$ is eliminated and x is replaced by t everywhere.
2. Then, the literals in $F[\bar{z}, x]$ are reordered as follows:
   (a) The lower bounds on x precede the upper bounds on x which precede the disequalities:
   $F[\bar{z}, x] = t_1 \lhd_1 x \wedge \ldots \wedge t_n \lhd_n x \wedge x \blacktriangleleft_1 s_1 \wedge \ldots \wedge x \blacktriangleleft_m s_m \wedge x \not\simeq u_1 \wedge \ldots \wedge x \not\simeq u_p$
   where $\forall i,\ 1 \leq i \leq n,\ \lhd_i \in \{<, \leq\}$ and $\forall j,\ 1 \leq j \leq m,\ \blacktriangleleft_j \in \{<, \leq\}$;
   (b) The lower bound literals are ordered in non-increasing order of the evaluation of the bounds in $\mathcal{M}$, and the upper bound literals are ordered in non-decreasing order of the evaluation of the bounds in $\mathcal{M}$:
   $F[\bar{z}, x] = t_1 \lhd_1 x \wedge \ldots \wedge t_n \lhd_n x \wedge x \blacktriangleleft_1 s_1 \wedge \ldots \wedge x \blacktriangleleft_m s_m \wedge x \not\simeq u_1 \wedge \ldots \wedge x \not\simeq u_p$
   where
   - $\forall i,\ 1 \leq i \leq n,\ \mathcal{M}(t_i) \geq \mathcal{M}(t_{i+1})$, and if $\mathcal{M}(t_i) = \mathcal{M}(t_{i+1})$ and $\lhd_i \neq \lhd_{i+1}$ then $\lhd_i$ is $<$ and $\lhd_{i+1}$ is $\leq$; and
   - $\forall j,\ 1 \leq j \leq m,\ \mathcal{M}(s_j) \leq \mathcal{M}(s_{j+1})$, and if $\mathcal{M}(s_j) = \mathcal{M}(s_{j+1})$ and $\blacktriangleleft_j \neq \blacktriangleleft_{j+1}$ then $\blacktriangleleft_j$ is $<$ and $\blacktriangleleft_{j+1}$ is $\leq$.
3. At this stage, $\mathsf{MBO}_{\mathsf{LRA}}(F[\bar{z}, x], x, \mathcal{M})$ is defined as follows:

$$\mathsf{MBO}_{\mathsf{LRA}}(F[\bar{z}, x], x, \mathcal{M}) = \begin{cases} s_1 \not\simeq u_k \vee u_k \not\simeq t_1 & \text{if } \mathcal{M} \models t_1 \lhd s_1, \\ t_1 \lhd s_1 & \text{otherwise,} \end{cases}$$

   where $1 \leq k \leq p$ and $\lhd$ is $<$ iff either $\lhd_1$ or $\blacktriangleleft_1$ is $<$. The rationale for this definition is the following. If $\mathcal{M} \models t_1 \lhd s_1$, it means that there exist values $q \in \mathbb{Q}$ for x that satisfy all lower bounds on x and all upper bounds on x in $F[\bar{z}, x]$. Since $\mathcal{M} \not\models \exists x. F[\bar{z}, x]$, it must be that for all such values q there exists a k, $1 \leq k \leq p$, such that $q = \mathcal{M}(u_k)$. The number of disequalities in $F[\bar{z}, x]$ is finite, whereas there are infinitely many rational numbers in the interval between $\mathcal{M}(t_1)$ and $\mathcal{M}(s_1)$. Therefore, it must be that $\mathcal{M}(t_1) = \mathcal{M}(s_1)$, both $\lhd_1$ and $\blacktriangleleft_1$ are $\leq$, and there exists a k, $1 \leq k \leq p$, such that $\mathcal{M}(u_k) = \mathcal{M}(t_1)$, so that $s_1 \simeq u_k \wedge u_k \simeq t_1$ is true in $\mathcal{M}$. Since $\mathsf{MBO}_{\mathsf{LRA}}(F[\bar{z}, x], x, \mathcal{M})$ is required to be false in $\mathcal{M}$, it is defined to be the negation of $s_1 \simeq u_k \wedge u_k \simeq t_1$. If $\mathcal{M} \not\models t_1 \lhd s_1$, the literal $t_1 \lhd s_1$ itself suffices.

The next theorem shows that $\mathsf{MBO}_{\mathsf{LRA}}$ satisfies the desiderata.

Theorem 9. *For all* LRA*-formulas $\exists x. F[\bar{z}, x]$, where $F[\bar{z}, x]$ is a quantifier-free conjunction of literals where x occurs in each literal, and all extensions $\mathcal{M}$ of $\mathcal{M}_0$ to $\bar{z}$ such that $\mathcal{M} \not\models \exists x. F[\bar{z}, x]$, the call $\mathsf{MBO}_{\mathsf{LRA}}(F[\bar{z}, x], x, \mathcal{M})$ returns a quantifier-free formula $O_{\mathsf{LRA}}[\bar{z}]$ such that $\mathcal{M} \not\models O_{\mathsf{LRA}}[\bar{z}]$ and $\mathsf{LRA} \models (\exists x. F[\bar{z}, x]) \Rightarrow O_{\mathsf{LRA}}[\bar{z}]$. Furthermore, the $\mathsf{MBO}_{\mathsf{LRA}}$ function has finite basis.*

Proof. For the first claim, we distinguish three cases, depending on whether $O_{\mathsf{LRA}}[\bar{z}]$ is the formula produced at Step (1), the formula produced in the first case of Step (3), or the formula produced in the second case of Step (3) of the above construction:

- $O_{\mathsf{LRA}}[\bar{z}]$ is the conjunction produced at Step (1)
  We show that $\mathcal{M} \not\models O_{\mathsf{LRA}}[\bar{z}]$. By hypothesis, $\mathcal{M} \not\models \exists x.F[\bar{z}, x]$. By way of contradiction, assume that $\mathcal{M} \models O_{\mathsf{LRA}}[\bar{z}]$. Since $F[\bar{z}, x]$ differs from $O_{\mathsf{LRA}}[\bar{z}]$ only by the addition of the literal $x \simeq t$ and the abstraction that replaces occurrences of term t with variable x, it suffices to interpret x as $\mathcal{M}(t)$ to have that $\mathcal{M} \models \exists x.F[\bar{z}, x]$, contradicting the hypothesis.
  We show $\mathsf{LRA} \models (\exists x.F[\bar{z}, x]) \Rightarrow O_{\mathsf{LRA}}[\bar{z}]$. We need to show that for all extensions $\mathcal{M}'$ of $\mathcal{M}_0$ to $\bar{z}$ such that $\mathcal{M}' \models \exists x.F[\bar{z}, x]$, it is also the case that $\mathcal{M}' \models O_{\mathsf{LRA}}[\bar{z}]$. Assume that $\mathcal{M}' \models \exists x.F[\bar{z}, x]$. This means that there exists a value $q \in \mathbb{Q}$ for x such that $\mathcal{M}' \models F[\bar{z}, x]$ with $\mathcal{M}'(x) = q$. Since the literal $x \simeq t$ (where x does not occur in t) is in the conjunction $F[\bar{z}, x]$, this means that $\mathcal{M}'(t) = q$. For all literals l other than $x \simeq t$ in $F[\bar{z}, x]$, $\mathcal{M}' \models l$ with $\mathcal{M}'(x) = q = \mathcal{M}'(t)$. Therefore, for all such literals l we have $\mathcal{M}' \models l\{x \leftarrow t\}$, and hence $\mathcal{M}' \models O_{\mathsf{LRA}}[\bar{z}]$.
- $O_{\mathsf{LRA}}[\bar{z}]$ is the disjunction $s_1 \not\simeq u_k \vee u_k \not\simeq t_1$ produced in the first case of Step (3)
  For $\mathcal{M} \not\models O_{\mathsf{LRA}}[\bar{z}]$, we have $\mathcal{M} \models s_1 \simeq u_k \wedge u_k \simeq t_1$ by construction, and hence $\mathcal{M} \not\models s_1 \not\simeq u_k \vee u_k \not\simeq t_1$ as desired.
  We show $\mathsf{LRA} \models (\exists x.F[\bar{z}, x]) \Rightarrow O_{\mathsf{LRA}}[\bar{z}]$. We need to show that for all extensions $\mathcal{M}'$ of $\mathcal{M}_0$ to $\bar{z}$ such that $\mathcal{M}' \models \exists x.F[\bar{z}, x]$, it is also the case that $\mathcal{M}' \models O_{\mathsf{LRA}}[\bar{z}]$. Assume that $\mathcal{M}' \models \exists x.F[\bar{z}, x]$. This means that there exists a value $q \in \mathbb{Q}$ for x such that $\mathcal{M}' \models F[\bar{z}, x]$ with $\mathcal{M}'(x) = q$. In particular, $\mathcal{M}' \models t_1 \lhd_1 x$, $\mathcal{M}' \models x \blacktriangleleft_1 s_1$, and $\mathcal{M}' \models x \not\simeq u_k$ with $\mathcal{M}'(x) = q$. The latter means that $\mathcal{M}'(x) \neq \mathcal{M}'(u_k)$. It follows that $\mathcal{M}' \models t_1 \lhd s_1$, where $\lhd$ is $<$ iff either $\lhd_1$ or $\blacktriangleleft_1$ is $<$, because $t_1 \lhd s_1$ is a sound consequence of $t_1 \lhd_1 x$ and $x \blacktriangleleft_1 s_1$ by FM-resolution. Then there are two cases: either $\mathcal{M}'(t_1) < \mathcal{M}'(s_1)$ or $\mathcal{M}'(t_1) = \mathcal{M}'(s_1)$. If $\mathcal{M}'(t_1) < \mathcal{M}'(s_1)$, then $\mathcal{M}' \not\models s_1 \simeq u_k \wedge u_k \simeq t_1$ and hence $\mathcal{M}' \models s_1 \not\simeq u_k \vee u_k \not\simeq t_1$ as desired. If $\mathcal{M}'(t_1) = \mathcal{M}'(s_1)$, we have $\mathcal{M}'(t_1) = \mathcal{M}'(s_1) = \mathcal{M}'(x)$, but $\mathcal{M}'(x) \neq \mathcal{M}'(u_k)$, so that again $\mathcal{M}' \not\models s_1 \simeq u_k \wedge u_k \simeq t_1$ and hence $\mathcal{M}' \models s_1 \not\simeq u_k \vee u_k \not\simeq t_1$ as desired.
- $O_{\mathsf{LRA}}[\bar{z}]$ is the literal $t_1 \lhd s_1$ produced in the second case of Step (3)
  $\mathcal{M} \not\models O_{\mathsf{LRA}}[\bar{z}]$ holds by construction.
  Also $\mathsf{LRA} \models (\exists x.F[\bar{z}, x]) \Rightarrow O_{\mathsf{LRA}}[\bar{z}]$ holds: for all extensions $\mathcal{M}'$ of $\mathcal{M}_0$ to $\bar{z}$ such that $\mathcal{M}' \models \exists x.F[\bar{z}, x]$, we have $\mathcal{M}' \models t_1 \lhd_1 x$ and $\mathcal{M}' \models x \blacktriangleleft_1 s_1$, so that $\mathcal{M}' \models t_1 \lhd s_1$, because $t_1 \lhd s_1$ is a sound consequence of $t_1 \lhd_1 x$ and $x \blacktriangleleft_1 s_1$ by FM-resolution.

For the second claim, the proof is the same as in the proof of Theorem 8. $\square$

7 The YicesQS Solver and Experimental Results

The OptiQSMA algorithm is implemented in the YicesQS solver built on top of the Yices 2 solver.[2] Therefore, YicesQS equips Yices 2 with reasoning about quantifiers in complete theories. This is unrelated to the support that Yices 2 offers towards reasoning about quantifiers in the theory of equality and free symbols (known as EUF for Equality with Uninterpreted Functions or UF for Uninterpreted Functions).

YicesQS relies on Yices 2 for concrete instances of the MBO and MBU abstract functions invoked by OptiQSMA. For MBO, model-based over-approximation is available as *model interpolation* from Yices's MCSAT [16] solver for quantifier-free formulas,

[2]See https://github.com/disteph/yicesQS for YicesQS and https://yices.csl.sri.com/ for Yices 2.

Table 1 Table for BV at SMT-COMP 2022

Solver	Solved	Time (s)
CVC5	854/970	25,584
Q3B	835/970	13,510
Z3	775/970	7,712
Bitwuzla	759/970	15,572
Q3B-pBDD	754/970	15,553
YicesQS	708/970	3,862
Ultimate Eliminator	304/970	4,204

including theory-specific techniques for bitvectors (BV) [21] and arithmetic [24]. The latter are based on the NLSAT procedure [23] and ultimately on *cylindrical algebraic decomposition* (CAD) [12, 13]. For MBU, basic model-based under-approximation is done as *model-guided generalization* [18] and improved with *model-based projection* for arithmetic, and *invertibility conditions* [35], including ϵ-terms, for BV. The model-based projection for arithmetic in YicesQS is also based on CAD.

In the 2022 SMT competition,[3] YicesQS entered the single-query, non-incremental tracks of BV, LRA, LIA, NRA, and NIA (Nonlinear Integer Arithmetic). The experiments were run on the Starexec cluster with a 20 minutes timeout per benchmark and 60GB of memory. The benchmarks were a subset of the SMT-LIB collection. The results presented below were computed by running the competition script `join.sh` on the raw data from StarExec,[4] sorting the data, and producing the plots that are available also online.[5] A description of the participating solvers can be found on the competition website.[6]

Table 1 and Figure 4 show the results for BV, where YicesQS solved quickly a high number of benchmarks (compared for example with CVC5), but was not outstanding, possibly because the 2022 version of YicesQS made a limited use of invertibility conditions for model interpolation.

Table 2 Table for LRA at SMT-COMP 2022

Solver	Solved	Time (s)
YicesQS	1003/1003	414
Z3 2021	948/1003	41,068
Z3	936/1003	41,240
Ultimate Eliminator	847/1003	16,136
CVC5	834/1003	21,197
Vampire	484/1003	45,326
SMTInterpol	164/1003	2,584

Tables 2, 3, and 4 show the results for the four arithmetics in terms of number of solved instances and time to solve them for each theory and solver. Figures 5, 6, 7,

[3]See https://smt-comp.github.io/2022/.
[4]See https://github.com/SMT-COMP/smt-comp/tree/master/2022/results.
[5]See http://www.csl.sri.com/users/sgl/Work/Cade2023-data/index.html.
[6]See https://smt-comp.github.io/2022/participants.html.

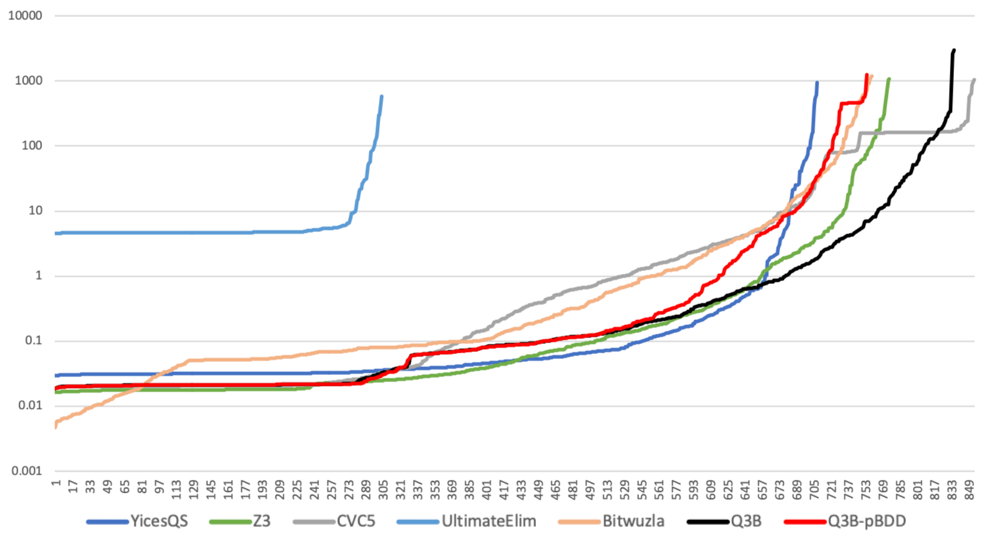

Fig. 4 Plots for the theory of bitvectors (BV) at SMT-COMP 2022

Table 3 Table for NRA and NIA at SMT-COMP 2022

	NRA		NIA	
Solver	Solved	Time (s)	Solved	Time (s)
YicesQS	94/99	165	80/108	290
Z3 2021	94/99	315	87/208	53
Z3	90/99	294	88/208	317
CVC5	86/99	672	190/208	3,642
Vampire	83/99	73	66/208	13,744
Ultimate Eliminator	6/99	33	129/208	701

and 8 show the plots for the four arithmetics. In the plots, each color corresponds to a solver and point (x, y) of that color means that the xth fastest-solved benchmark was solved by that solver in time y (log scale). The 2021 version of Z3 is included because in some of these theories it performed slightly better than the 2022 version.

The theory where YicesQS performed best is LRA: it was the only solver to solve all 1003 benchmarks. Z3 2021 was second best, solving 948 benchmarks with a total runtime about 100 times higher. YicesQS has neither a special treatment (e.g., simplex-based) of linear problems, nor integer-specific techniques: it relies on CAD-based techniques for MBU and MBO also for integer problems. Thus, it is somewhat average on LIA and NIA. The NRA and NIA theories are undecidable (NRA due to division by 0) and hence they lie outside of the theoretical framework of QSMA. YicesQS answers should still be correct, but termination can be lost. With Z3 being a non-competing participant in the SMT 2022 competition, YicesQS came second for *Largest Contribution* (single queries), because of its overall performance in the four arithmetics, where

Table 4 Table for LIA at SMT-COMP 2022

Solver	Solved	Time (s)
Z3	300/300	11
CVC5	300/300	78
Z3 2021	292/300	10
Ultimate Eliminator	230/300	11,789
YicesQS	182/300	750
Vampire	157/300	985
SMTInterpol	97/300	134
VeriT	75/300	1

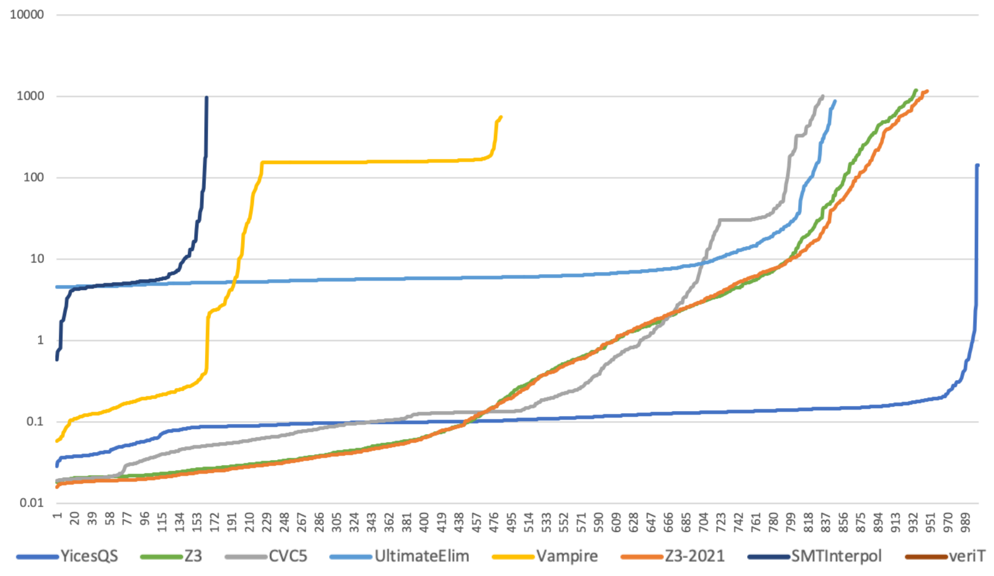

Fig. 5 Plots for LRA at SMT-COMP 2022

it also came first for satisfiable instances (SAT performance) and in the 24 s (instead of 20 min) timeout setup (24 s performance).

In the 2023 and 2024 SMT competitions,[7] YicesQS again entered the single-query, non-incremental tracks of BV, LRA, LIA, NRA, and NIA. In 2023 the results were similar to those of 2022: in the Arith division, which includes LRA, LIA, NRA, and NIA, YicesQS was the first solver for SAT performance and 24 s performance, and it was among the first three solvers in all five columns (sequential performance, parallel performance, SAT performance, UNSAT performance, 24 s performance). In 2024 YicesQS was either the first or the second solver in all five columns in the Arith division, winning again in the SAT performance and 24 s performance columns. Furthermore, in 2024 YicesQS improved significantly its performance in the BV division,

[7]See https://smt-comp.github.io/2023/ and https://smt-comp.github.io/2024/.

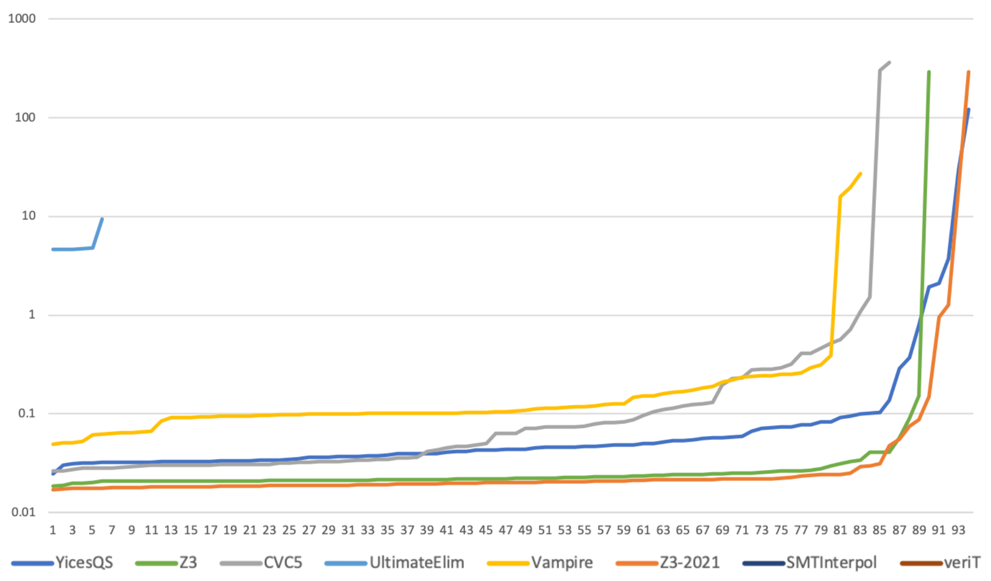

Fig. 6 Plots for NRA at SMT-COMP 2022

Table 5 Placement of YicesQS in the SAT and UNSAT performance columns

	SMT-COMP 2022		SMT-COMP 2023		SMT-COMP 2024	
Division	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT
Arith	I	II	I	II	I	II
Bitvec	IV	V	III	IV	III	II

where it was among the first three solvers in all five columns, winning in the 24 s performance column.

It is interesting to know whether a solver implementing a new algorithm performs better on satisfiable or unsatisfiable instances. Table 5 reports the placement of YicesQS in the SAT and UNSAT performance columns over three years. The QSMA and OptiQSMA algorithms are not geared for either class of problems, and indeed the data do not indicate a marked preference of YicesQS for either satisfiable or unsatisfiable instances.

8 Discussion

We presented the QSMA algorithm for *quantified satisfiability modulo theory and assignment*, its optimization OptiQSMA, and experimental results with the YicesQS solver that implements the OptiQSMA algorithm.

The QSMA algorithm performs a recursive descent on the structure of the input formula represented as a QSMA-tree. At each node, it removes one block of quantifiers

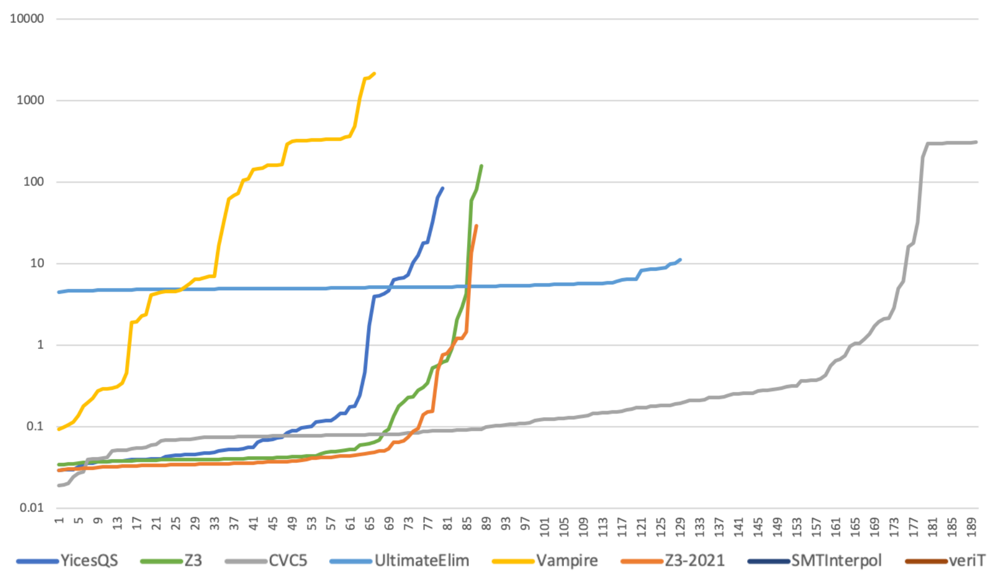

Fig. 7 Plots for NIA at SMT-COMP 2022

and proposes an assignment of values to the first-order variables thus freed. Technically, an assignment is an extension of the unique model of the theory. Then, QSMA recurses on the subtrees representing the quantified subformulas. Each quantified subformula is abstracted as a Boolean proxy variable. An extension that assigns values to freed first-order variables and Boolean proxy variables is generated by a call to an underlying solver for quantifier-free satisfiability modulo theory and assignment. QSMA passes to this solver a formula whose satisfaction is a necessary condition for success (i.e., satisfiability). Thus, if the underlying solver fails to find an extension of the current assignment that satisfies this necessary condition, the QSMA algorithm can safely return *false*. Otherwise, since the formula given to the underlying solver is a necessary, but not sufficient, condition for success, the QSMA algorithm recurses on the subtrees, and it return *true* if all these recursive calls succeeds.

A key feature of the QSMA algorithm is that it maintains *under-approximations* and *over-approximations* of all subformulas in the problem. These approximations are *model-based*, because they are computed relative to the current extension. A *model-based over-approximation* (MBO) is an implied formula that is false in the given extension, whereas a *model-based under-approximation* (MBU) is an implicant that is true in the given extension. The QSMA algorithm makes MBO's progressively stronger and MBU's progressively weaker. The strengthening results from conjunction with an MBO of the formula that represents a necessary condition for success. The weakening results from disjunction with an MBU of another formula, which represents a sufficient condition for success. By progressively deflating the MBO model set and inflating the MBU model set (see Figure 3), the QSMA algorithm guides the search towards a solution (i.e., a model of the given formula) or the discovery that none exists.

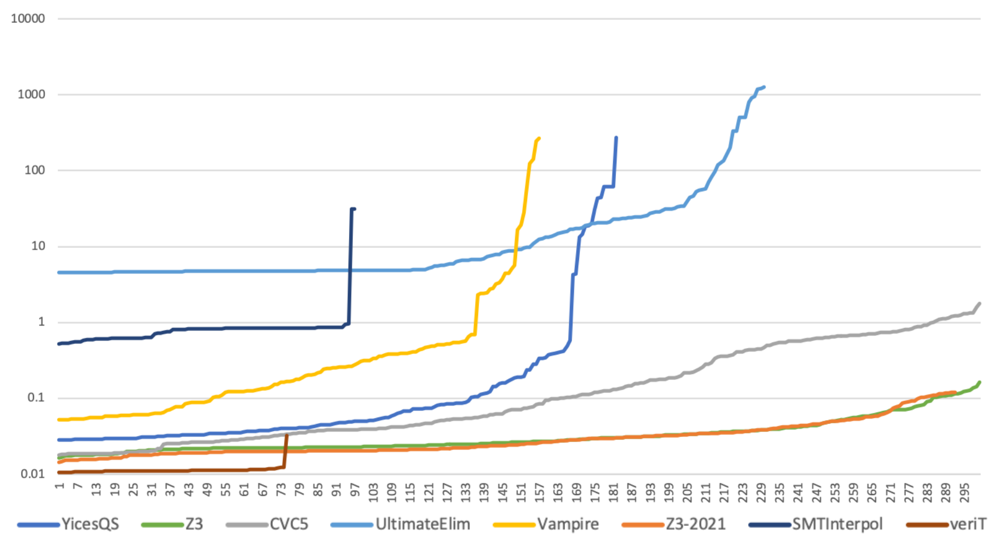

Fig. 8 Plots for LIA at SMT-COMP 2022

The OptiQSMA algorithm optimizes the QSMA algorithm in two ways. First, it optimizes the recursive descent, by a *look-ahead mechanism* that reduces the number of recursive calls. It is not necessary to recurse on all the children of a node. It suffices to recurse on the nodes on a further away frontier, identified based on the assignment to proxy variables. For this optimization the OptiQSMA algorithm passes to the underlying solver a richer formula, called *look-ahead formula*, which covers the subtree of the current node down to the frontier. In this manner the number of recursive calls is reduced, and each call to the underlying solver does more work, assigning values to more variables or detecting a failure sooner. The second optimization consists of sparing the effort of maintaining MBO's for all subformulas. While MBU's are maintained, MBO's are computed, but not saved: they are only used in the formula that represents a sufficient condition for success. We proved the total correctness of both QSMA and OptiQSMA. Termination rests on the assumption that the MBU and MBO functions used to compute MBU's and MBO's, respectively, have *finite basis*, which means that only finitely many formulas can be produced for a given formula.

8.1 Comparison with Related Work

Under-approximations and over-approximations of formulas appear in various contexts in automated reasoning. Under and over-approximations of axioms were combined to guide the search in theorem proving [32]. Abstract interpretation uses *widening*, which is an over-approximation mechanism, but does not have an under-approximation counterpart (e.g., [11, Ch. 12] for a reasoning-oriented presentation). The model-checking method of [26] generates under and over-approximations of procedure summaries. We used under and over-approximations of formulas for quantified SMT. Most existing approaches to quantified SMT are *instantiation-based*, including: *E-matching* [14, 17, 20], and *model-based* [19, 41], *conflict-based* [40], *counterexample-guided* [39], *enumeration-based* [22, 38], and *syntax-guided* [36] instantiation.

A basic instantiation-based approach interfaces an SMT solver for the quantifier-free fragment of the theory with an instantiation layer. Suppose that the input quantified formula φ is a set of clauses. The instantiation layer generates a finite set of instances of the clauses, which is given to the SMT solver: if it is unsatisfiable, φ is unsatisfiable; if it is satisfiable, more instances need to be generated to discover unsatisfiability. The various approaches differ in how the instances are generated. A basic instantiation-based approach is geared toward discovering unsatisfiability.

Counterexample-guided (CEG) instantiation is the method implemented in the CVC5 solver to handle quantifiers in LRA, LIA and their combination [39]. CEG instantiation applies to formulas with arbitrary alternations of quantifiers. For LRA, CEG instantiation employs substitutions that replace variables with lower bounds or upper bounds, including *virtual term substitutions* (i.e., substitutions that replace variables with symbols, such as δ for an infinitesimal positive value, that are not terms in the signature of the theory). CEG instantiation determines that the input quantified formula φ is satisfiable, by discovering a satisfiable set S of instances of φ such that $S \models \varphi$ in the theory. Whenever a set S of instances is found to be satisfiable, the SMT solver is applied to $S \cup \{\neg\varphi\}$, and if $S \cup \{\neg\varphi\}$ is found unsatisfiable, φ is reported to be satisfiable. The set S can be seen as an MBU of φ, because S is quantifier-free, $S \models \varphi$ in the theory, and S is true in the current model in the solver, namely the one produced when discovering the satisfiability of S. However, the current model in the solver is represented by a Boolean assignment, whereas the MBU concept used in QSMA is more general, because it is relative to an extension that contains *both Boolean and first-order assignments.*

The QSMA algorithm is not instantiation based; it does not generate instances of the input formula φ, by applying substitutions that replace quantified variables with terms; and it does not call iteratively the underlying quantifier-free SMT solver on sets of instances. QSMA works with φ (and its subformulas) and an extension $\mathcal{M}$ of the unique model $\mathcal{M}_0$ of the theory. The extension $\mathcal{M}$ is written as an assignment that maps variables to values, not terms. Values are special constants introduced by a conservative extension of the theory to name the elements of the domain of $\mathcal{M}_0$. The extension $\mathcal{M}$ contains both Boolean and first-order assignments. QSMA is not geared for either unsatisfiability or satisfiability.

Two methods that employ model-based approximation for quantified SMT are the *Two-Solver procedure* [18], implemented for LRA in the Yices solver, and the *QSAT algorithm* [3], implemented in the Z3 solver, where it is the default solver for LIA, LRA, and NRA. In terms of the class of input formulas, the Two-Solver procedure accepts prenex normal form formulas with $\exists\forall$ prefix, the QSAT algorithm accepts prenex normal form formulas with $(\exists\forall)^+$ prefix, and the QSMA algorithm accepts arbitrary formulas with quantifiers in arbitrary positions. Therefore, QSMA is the most general of the three in this regard.

In terms of underlying functionalities, the Two-Solver procedure invokes an $\exists$-solver and a $\forall$-solver, and it employs *model-guided generalization* [18], based on virtual term substitutions [31, 42, 43], as model-based under-approximation for LRA. The implementation in Yices featured also FM-resolution, using it only when it is inexpensive in terms of number of generated atoms. The QSAT algorithm is formulated as a game between an $\exists$-player and a $\forall$-player, it invokes a solver for quantifier-free satisfiability, and it employs *model-based projection* [3, 25, 26] as model-based under-approximation. Model-based projection for LRA [26] uses *virtual term substitutions* following the Loos-Weispfenning method [31] for quantifier elimination (QE) as presented in [37]. Neither the Two-Solver procedure nor the QSAT algorithm use model-based over-approximation. Using both MBU's and MBO's, in order to corner a solution from below and from above, is an essential and novel characteristic of QSMA.

The QSAT algorithm is relatively closer to QSMA, because QSAT requires the underlying solver to support *unsatisfiable cores*. For a quantified formula φ and a Boolean assignment $\mathcal{M}$, an unsatisfiable core in QSAT is a quantifier-free formula $G[\bar{z}]$ such that $\varphi \wedge G[\bar{z}]$ is unsatisfiable, $\mathcal{M} \models G[\bar{z}]$, and all non-variable symbols occurring in $G[\bar{z}]$ occur in $\mathcal{M}$. Clearly, $\neg G[\bar{z}]$ is an MBO of φ, because the unsatisfiability of $\varphi \wedge G[\bar{z}]$ implies that $\varphi \Rightarrow \neg G[\bar{z}]$ is valid in the theory, and $\mathcal{M} \models G[\bar{z}]$ implies $\mathcal{M} \not\models \neg G[\bar{z}]$. An MBO function can produce $O[\bar{z}] = \neg G[\bar{z}]$, for $G[\bar{z}]$ an unsatisfiable core in the QSAT sense. However, the concept of MBO in QSMA generalizes the notion of unsatisfiable core with theory-specific reasoning, when working modulo an assignment $\mathcal{M}$ that contains both first-order and Boolean assignments. Reasoning modulo such an assignment is another essential and innovative feature of QSMA. It is unclear whether the combination of unsatisfiable cores and theory-specific model-based under-approximation can emulate model-based over-approximation or provide the same benefits. In summary, the differences include:

1. QSAT accepts formulas in prenex normal form with an $(\exists\forall)^+$ prefix, whereas QSMA accepts arbitrary formulas with quantifiers in arbitrary positions;
2. QSMA uses MBU's and MBO's, whereas QSAT uses MBU's and unsatisfiable cores;
3. QSMA uses MBU's and MBO's where the model is an assignment that contains both first-order and Boolean assignments, whereas an unsatisfiable core is (the negation of) an MBO where the model is a Boolean assignment;
4. QSAT is iterative, whereas QSMA is recursive, but OptiQSMA limits the recursion.

8.2 Directions for future work

The framework of definitions and the QSMA and OptiQSMA algorithms assume that the assignment to Boolean proxy variables is total. This is apparent beginning with Definition 3, where the Boolean assignment determines the mission of the algorithm on a subformula (whether it should try to show it true or false). However, the need for a total assignment appears wasteful, for example, whenever the topmost connective in a formula at a node is a disjunction. Therefore, a first direction for future work is to generalize framework and algorithms to assignments that include *undefined* Boolean proxy variables. In the practice it is common that a call to an underlying solver returns a partial assignment, and the generalization to partial assignments is simple for some aspects (e.g., skip the recursive call on a child node b such that $b.p$ is undefined in Algorithm 2). However, the theoretical framework may have to be redone in a three-valued logic, and the generalization may not be as simple for other ingredients, such as formulas L and L' in both algorithms, look-ahead formula, no-alternation and first-alternation node sets, which are all written under the assumption that the assignment is total, so that the truth value on an arc works as a signal that tells what to do.

The conjecture that if a theory has MBU and MBO functions with finite basis then the theory admits quantifier elimination (QE) is open. If this conjecture were true, it may be possible to turn QSMA into a QE algorithm, by ensuring that it reaches a stage where $n.U \Leftrightarrow n.\psi \Leftrightarrow n.O$ for all nodes n in the QSMA-tree. Even if the conjecture were true, the QSMA algorithm could still be a better performing way of eliminating quantifiers towards deciding satisfiability. Indeed, QSMA does not apply QE upfront, and $n.U \Leftrightarrow n.\psi \Leftrightarrow n.O$ is not a condition for termination on the QSMA-tree with n as root. QSMA removes quantifiers lazily until satisfiability can be decided.

The QSMA algorithm applies to a complete theory with unique model. In this article, we considered the four arithmetics and the theory of bitvectors for implementation and experiments, and LRA for examples of MBU and MBO functions. In future work, we may consider applying or extending QSMA to other theories (e.g., floating point arithmetic), or integrating QSMA with the CDSAT (*Conflict-Driven Satisfiability*) method for reasoning in a union of theories [6–8]. The latter direction may require to go beyond the restriction to theories with a unique model. Dropping this assumption would be a generalization also in the single theory case. As most known MBU and MBO functions are for single theories, another issue is how to get MBU and MBO functions for a union of theories from such functions for the member theories. The interplay between QSMA's recursive descent over the formula and CDSAT's conflict-driven search would also need to be understood.

Possible directions of future work on YicesQS include augmenting its reasoning capabilities on the integers, since YicesQS lacks integer-specific techniques, and uses the MBU and MBO functions for LRA and NRA also for LIA and NIA, respectively. Although YicesQS performs best in LRA, also the linear case may be susceptible of improvement, because YicesQS does not have a simplex-based treatment of linear problems. The recent *FMplex algorithm* for LRA can be used for QE and for SMT solving [34]. For the latter purpose FMplex hybridates FM-resolution and simplex, reducing the complexity of a systematic application of FM-resolution from double to single exponential in the worst case. FMplex may be useful for the design of MBU and MBO functions for LRA and for the treatment of linear problems in YicesQS.

Acknowledgements. Part of this work was done while the first and third authors were visiting the Computer Science Laboratory of SRI International, whose support is appreciated. This research was supported in part by an Amazon Research Award to the first author, and NSF grants CCF-1816936 and CCF-1817204 to the second author. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

References

[1] Althaus, E., Kruglov, E., Weidenbach, C.: Superposition modulo linear arithmetic SUP(LA). In: Ghilardi, S., Sebastiani, R. (eds.) Proc. FroCoS-7. LNAI, vol. 5749, pp. 84–99. Springer, Berlin (2009). https://doi.org/10.1007/978-3-642-04222-5_5

[2] Baumgartner, P., Waldmann, U.: Hierarchic superposition with weak abstraction. In: Bonacina, M.P. (ed.) Proc. CADE-24. LNAI, vol. 7898, pp. 39–57. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38574-2_3

[3] Bjørner, N., Janota, M.: Playing with quantified satisfaction (Short paper). In: Fehnker, A., McIver, A., Sutcliffe, G., Voronkov, A. (eds.) Short Presentations at LPAR-20. EPiC Series in Computing, vol. 35, pp. 15–27. EasyChair, Manchester (2015)

[4] Bonacina, M.P.: Reasoning about quantifiers in SMT: the QSMA algorithm (Abstract). In: Nadel, A., Rozier, K.Y. (eds.) Proc. FMCAD-23, pp. 1–1. TU Wien Academic Press, Vienna (2023). https://doi.org/10.34727/2023/isbn.978-3-85448-060-0_1

[5] Bonacina, M.P., Plaisted, D.A.: Semantically-guided goal-sensitive reasoning: inference system and completeness. J. Autom. Reason. **59**(2), 165–218 (2017) https://doi.org/10.1007/s10817-016-9384-2

[6] Bonacina, M.P., Graham-Lengrand, S., Shankar, N.: Conflict-driven satisfiability for theory combination: transition system and completeness. J. Autom. Reason. **64**(3), 579–609 (2020) https://doi.org/10.1007/s10817-018-09510-y

[7] Bonacina, M.P., Graham-Lengrand, S., Shankar, N.: CDSAT for nondisjoint theories with shared predicates: arrays with abstract length. In: Hyvärinen, A., Déharbe, D. (eds.) Proc. SMT-20. CEUR Proceedings, vol. 3185, pp. 18–37. CEUR WS-org, Aachen (2022). https://ceur-ws.org/Vol-3185/

[8] Bonacina, M.P., Graham-Lengrand, S., Shankar, N.: Conflict-driven satisfiability for theory combination: lemmas, modules, and proofs. J. Autom. Reason. **66**(1), 43–91 (2022) https://doi.org/10.1007/s10817-021-09606-y

[9] Bonacina, M.P., Graham-Lengrand, S., Vauthier, C.: QSMA: a new algorithm for quantified satisfiability modulo theory and assignment. In: Pientka, B., Tinelli, C. (eds.) Proc. CADE-29. LNAI, vol. 14132, pp. 78–95. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38499-8_5

[10] Bonacina, M.P., Lynch, C.A., de Moura, L.: On deciding satisfiability by theorem proving with speculative inferences. J. Autom. Reason. **47**(2), 161–189 (2011) https://doi.org/10.1007/s10817-010-9213-y

[11] Bradley, A.R., Manna, Z.: The Calculus of Computation - Decision Procedures with Applications to Verification. Springer, Berlin (2007). https://doi.org/10.1007/978-3-540-74113-8

[12] Caviness, B.F., Johnson, J.R.: Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts and Monographs in Symbolic Computation. Springer, Wien (1998). https://doi.org/10.1007/978-3-7091-9459-1

[13] Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In: Brakhage, H. (ed.) Proc. 2nd GI Conf. on Automata Theory and Formal Languages. LNCS, vol. 33, pp. 134–183. Springer, Heidelberg (1975). https://doi.org/10.1007/3-540-07407-4_17

[14] de Moura, L., Bjørner, N.: Efficient E-matching for SMT-solvers. In: Pfenning, F. (ed.) Proc. CADE-21. LNAI, vol. 4603, pp. 183–198. Springer, Berlin (2007). https://doi.org/10.1007/978-3-540-73595-3_13

[15] de Moura, L., Bjørner, N.: Bugs, moles, and skeletons: symbolic reasoning for software development. In: Giesl, J., Hähnle, R. (eds.) Proc. IJCAR-5. LNAI, vol. 6173, pp. 400–411. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-14203-1_34

[16] de Moura, L., Jovanović, D.: A model-constructing satisfiability calculus. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) Proc. VMCAI-14. LNCS, vol. 7737, pp. 1–12. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35873-9_1

[17] Detlefs, D.L., Nelson, G., Saxe, J.B.: Simplify: a theorem prover for program checking. J. ACM **52**(3), 365–473 (2005) https://doi.org/10.1145/1066100.1066102

[18] Dutertre, B.: Solving exists/forall problems with Yices (2015). https://brunodutertre.github.io/

[19] Ge, Y., de Moura, L.: Complete instantiation for quantified formulas in satisfiability modulo theories. In: Bouajjani, A., Maler, O. (eds.) Proc. CAV-21. LNCS, vol. 5643, pp. 306–320. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_25

[20] Ge, Y., Barrett, C., Tinelli, C.: Solving quantified verification conditions using satisfiability modulo theories. In: Pfenning, F. (ed.) Proc. CADE-21. LNAI, vol. 4603, pp. 167–182. Springer, Berlin (2007). https://doi.org/10.1007/978-3-540-73595-3_12

[21] Graham-Lengrand, S., Jovanović, D., Dutertre, B.: Solving bitvectors with MCSAT: explanations from bits and pieces. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) Proc. IJCAR-10. LNAI, vol. 12166, pp. 103–121. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51074-9_7

[22] Janota, M., Barbosa, H., Fontaine, P., Reynolds, A.: Fair and adventurous enumeration of quantifier instantiations. In: Piskac, R., Whalen, M.W. (eds.) Proc. FMCAD-21, pp. 256–260. TU Wien Academic Press, Vienna (2021). https://doi.org/10.34727/2021/isbn.978-3-85448-046-4_35

[23] Jovanović, D., de Moura, L.: Solving non-linear arithmetic. In: Gramlich, B.,

Miller, D., Sattler, U. (eds.) Proc. IJCAR-6. LNAI, vol. 7364, pp. 339–354. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-31365-3_27

[24] Jovanović, D., Dutertre, B.: Interpolation and model checking for nonlinear arithmetic. In: Silva, A., Leino, K.R.M. (eds.) Proc. CAV-33. LNCS, vol. 12760, pp. 266–288. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81688-9_13

[25] Komuravelli, A., Gurfinkel, A., Chaki, S.: SMT-based model checking for recursive programs. In: Biere, A., Bloem, R. (eds.) Proc. CAV-26. LNCS, vol. 8559, pp. 17–34. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-319-08867-9_2

[26] Komuravelli, A., Gurfinkel, A., Chaki, S.: SMT-based model checking for recursive programs. Form. Methods Syst. Des. **48**(3), 175–205 (2016) https://doi.org/10.1007/s10703-016-0249-4

[27] Komuravelli, A., Bjørner, N., Gurfinkel, A., McMillan, K.L.: Compositional verification of procedural programs using Horn clauses over integers and arrays. In: Kaivola, R., Wahl, T. (eds.) Proc. FMCAD 2015, pp. 89–96. FMCAD Inc., Austin (2015). https://doi.org/10.5555/2893529.2893548

[28] Korovin, K., Voronkov, A.: Integrating linear arithmetic into superposition calculus. In: Duparc, J., Henzinger, T.A. (eds.) Proc. CSL-16. LNCS, vol. 4646, pp. 223–237. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74915-8_19

[29] Kroening, D., Strichman, O.: Decision Procedures - An Algorithmic Point of View. Texts in Theoretical Computer Science. Springer, Berlin (2008)

[30] Lassez, J.-L., Maher, M.J.: On Fourier's algorithm for linear arithmetic constraints. J. Autom. Reason. **9**, 373–379 (1992) https://doi.org/10.1007/BF00245296

[31] Loos, R., Weispfenning, V.: Applying linear quantifier elimination. Comput. J. **36**(5), 450–462 (1993) https://doi.org/10.1093/comjnl/36.5.450

[32] Lopez Hernandez, J., Korovin, K.: An abstraction refinement framework for reasoning with large theories. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) Proc. IJCAR-9. LNAI, vol. 10900, pp. 663–679. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94205-6_43

[33] Moskal, M.: Fx7 or in software, it is all about quantifiers. System Descriptions at SMT-COMP (2007). http://smtcomp.cs.uiowa.edu/2007/descriptions/fx7.pdf

[34] Nalbach, J., Promies, V., Ábrahám, E., Kobialka, P.: FMplex: A novel method for solving linear real arithmetic problems. In: Achilleos, A., Monica, D.D. (eds.) Proc. GandALF-14. EPTCS, vol. 390, pp. 16–32 (2023). https://doi.org/10.4204/EPTCS.390.2

[35] Niemetz, A., Preiner, M., Reynolds, A., Barrett, C., Tinelli, C.: Solving quantified bit-vectors using invertibility conditions. In: Chockler, H., Weissenbacher, G. (eds.) Proc. CAV-30. LNCS, vol. 10982, pp. 236–255. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-96142-2_16

[36] Niemetz, A., Preiner, M., Reynolds, A., Barrett, C., Tinelli, C.: Syntax-guided quantifier instantiation. In: Groote, J., Larsen, K. (eds.) Proc. TACAS-27. LNCS, vol. 12652, pp. 145–163. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72013-1_8

[37] Nipkow, T.: Linear quantifier elimination. J. Autom. Reason. **45**(2), 189–212 (2010) https://doi.org/10.1007/s10817-010-9183-0

[38] Reynolds, A., Barbosa, H., Fontaine, P.: Revisiting enumerative instantiation. In: Beyer, D., Huisman, M. (eds.) Proc. TACAS-24. LNCS, vol. 10806, pp. 112–131. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89963-3_7

[39] Reynolds, A., King, T., Kuncak, V.: Solving quantified linear arithmetic by counterexample-guided instantiation. Form. Methods Syst. Des. **51**, 500–532 (2017) https://doi.org/10.1007/s10703-017-0290-y

[40] Reynolds, A., Tinelli, C., de Moura, L.: Finding conflicting instances of quantified formulas in SMT. In: Claessen, K., Kuncak, V. (eds.) Proc. FMCAD 2014, pp. 195–202. FMCAD Inc., Austin (2014). https://doi.org/10.5555/2682923.2682957

[41] Reynolds, A., Tinelli, C., Goel, A., Krstić, S., Deters, M., Barrett, C.: Quantifier instantiation techniques for finite model finding in SMT. In: Bonacina, M.P. (ed.) Proc. CADE-24. LNAI, vol. 7898, pp. 377–391. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38574-2_26

[42] Weispfenning, V.: The complexity of linear problems in fields. J. Symb. Comput. **5**(1–2), 2–27 (1988) https://doi.org/10.1016/S0747-7171(88)80003-8

[43] Weispfenning, V.: Quantifier elimination for real algebra – the quadratic case and beyond. Appl. Algebra Eng. Commun. Comput. **8**(2), 85–101 (1997) https://doi.org/10.1007/s002000050055