Form Methods Syst Des (2017) 51:500–532
https://doi.org/10.1007/s10703-017-0290-y

Solving quantified linear arithmetic by counterexample-guided instantiation

Andrew Reynolds[1] · **Tim King**[3] · **Viktor Kuncak**[2]

Published online: 3 August 2017

Abstract This paper presents a framework to derive instantiation-based decision procedures for satisfiability of quantified formulas in first-order theories, including its correctness, implementation, and evaluation. Using this framework we derive decision procedures for linear real arithmetic and linear integer arithmetic formulas with one quantifier alternation. We discuss extensions of these techniques for handling mixed real and integer arithmetic, and to formulas with arbitrary quantifier alternations. For the latter, we use a novel strategy that handles quantified formulas that are not in prenex normal form, which has advantages with respect to existing approaches. All of these techniques can be integrated within the solving architecture used by typical SMT solvers. Experimental results on standardized benchmarks from model checking, static analysis, and synthesis show that our implementation in the SMT solver CVC4 outperforms existing tools for quantified linear arithmetic.

Keywords Satisfiability modulo theories · Quantifier elimination · Linear arithmetic · Quantifier Instantiation

1 Introduction

Among the biggest challenges in automated reasoning is efficient support for quantifiers in the presence of background theories. Quantifiers enable direct encoding of a number of problems of interest, including synthesis of software fragments from specifications [34,50,

✉ Andrew Reynolds
andrew.j.reynolds@gmail.com

Viktor Kuncak
viktor.kuncak@epfl.ch

1 Department of Computer Science, The University of Iowa, Iowa city, IA, USA

2 École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

3 Google, Mountain view, CA, USA

56], construction of transfer functions for program analysis [40], invariant inference [14,27], as well as analysis of properties that go beyond safety [10,11].

The most commonly used complete method for deciding constraints over quantified theories is *quantifier elimination* [29, Section 2.7]. Quantifier elimination algorithms typically solve a more general problem, of transforming arbitrary quantified formula with free variables into a theory-equivalent formula with no quantifiers.

However, depending on the particular variant of the language of constraints, performing actual quantifier elimination can have worse complexity than the decision problem [9], in part because it is required to give an answer on any formula, and the smallest formula resulting from quantifier elimination can be very large [64]. When the goal is to decide the satisfiability of quantified constraints, quantifier elimination may be doing unnecessary work. More importantly, procedures based on quantifier elimination often do not handle the underlying ground constraints in the most efficient way. Thus, quantifier elimination tends to be prohibitively expensive in practice. Recent work involving quantifier elimination [12, 41] has been motivated by avoiding worst-case performance by effectively computing an equisatisfiable set of ground formulas in a lazy fashion.

In the broader scope of automated theorem proving, it is often important to reason about formulas involving multiple theories, each of which may or may not support quantifier elimination. In practice, the goal is to obtain a framework for handling quantified formulas that is both complete for formulas belonging to decidable logics, and empirically effective when completeness guarantees are not known. To this end, modern SMT solvers most commonly use heuristic instantiation-based approaches [18], which are incomplete but work well in practice for undecidable fragments of first-order logic. A long term goal of this work is to capitalize both on recent advances in specialized techniques for quantified linear arithmetic [12,13,32,44], and recent advances in instantiation-based theorem proving for first-order logic [17,24,51].

In this paper, we introduce an approach for establishing the satisfiability of formulas in quantified linear arithmetic based on a new *quantifier instantiation* framework. The use of quantifier instantiation for this task is motivated by the following.

- Procedures based on lazy quantifier instantiation typically establish satisfiability much faster than their theoretical complexity.
- Using quantifier instantiation for decidable fragments enables a uniform integration and composition with existing instantiation-based techniques [17,18,51], which are widely used by modern SMT solvers.
- An important class of synthesis problems can be expressed as quantified formulas with one quantifier alternation. As shown in [50], solutions for these problems can be extracted from an unsatisfiable core of quantifier instantiations.

Related work Quantifier elimination has been used to, e.g., show decidability and classification of Boolean algebras [55,61], Presburger arithmetic [46], decidability of products [22,42], [39, Chapter 12], and algebraically closed fields [60]. The original result on decidability of Presburger arithmetic is by Presburger [46]. The space bound for Presburger arithmetic was shown in [23]. The matching lower and upper bounds for Presburger arithmetic were shown in [9], see also [33, Lecture 24]. An analysis parameterized by the number of quantifier alternations is presented in [48]. A mechanically verified quantifier elimination algorithm was developed by Nipkow [43].

An approach for lazy quantifier elimination for linear real arithmetic was developed by Monniaux [41]. Integration of linear quantifier elimination into the solving algorithm used by SMT solvers was developed in [12], though the presented integration is not model driven.

A lazy approach for quantifier elimination, which relies on an operation called model-based projection, has been developed in the context of SMT-based model checking [32], and can be used for extracting Skolem functions for simulation synthesis [21]. An efficient approach for quantified linear real arithmetic that is based on finding player strategies for establishing satisfiability is given in [20]. A recent approach involving quantified formulas with arbitrary alternations has been developed by Bjørner and Janota [13] for several background theories, which is closely related to the approach in this paper. By comparison, their approach is described in terms of a model-based projection operation whereas ours is based on instantiation. Additionally, their strategy for handling quantified formulas with multiple alternations assumes quantified formulas are in prenex form, whereas ours handles a more general grammar of quantified formulas that includes those not in prenex normal form. We will comment more on the technical differences between these two approaches in the later sections.

The most widely used techniques for quantifier instantiation in SMT were developed in [18], and later in [17,25], which primarily focused on quantified formulas with uninterpreted functions. Our approach for quantified linear arithmetic instantiates quantified formulas based on a lazy stream of candidate models, terminating when either it finds a finite set of instances are unsatisfiable, or discovers that the original formula is satisfiable. Other approaches in this spirit have been used for quantified Boolean formulas [31], quantified bit-vectors [66], the essentially uninterpreted fragment [26], and, more generally, theories having a locality property [5,30]; these works do not directly apply to quantified linear arithmetic. A recent approach for quantified formulas with one quantifier alternation has been developed in the SMT solver Yices [19]. The present paper builds upon our previous work for solving synthesis conjectures using quantifier instantiation in SMT [50], where an approach for quantified linear arithmetic was described without a specific method for selecting instances and without completeness guarantees. While the present paper focuses on linear arithmetic, where it outperforms existing approaches, we expect the presented framework to be relevant for other quantified theories. Among the examples of further decidable quantified constraints are quantified theories of term algebras [39, Chapter23], [38,57] and their extensions [15,35,52], feature trees [4,62], and monadic second-order theories [63].

Contributions This paper makes the following contributions. First, we define a general class of instantiation-based procedures for establishing the satisfiability of quantified formulas in Sect. 2. We demonstrate instances of the procedure are sound and complete for formulas over linear real arithmetic (LRA) and linear integer arithmetic (LIA) with one quantifier alternation in Sects. 3 and 4, two quantified fragments for which many current SMT solvers do not have efficient support for. We describe how these two procedures can be combined for mixed real and integer arithmetic (LIRA) in Sect. 5, although completeness for this fragment is left for future work. We show how our procedure can be integrated into the solving architecture used by SMT solvers and how the procedure can be used for solving formulas with arbitrary quantifier alternation in Sect. 6. A key feature of our strategy for handling quantifier alternations is that we do not impose the restriction that quantified formulas must be in prenex normal form. We show how instantiation procedures can be used in part for solving synthesis problems in Sect. 7. Our approach is sound and complete for quantified linear arithmetic and is based purely on quantifier instantiation, which has the advantage of being composable with existing techniques and whose soundness is straightforward to verify. Section 8 gives experimental results for an implementation of the procedures for LIA and LRA in the SMT solver CVC4, which in addition to having the aforementioned advantages, outperforms state-of-the-art SMT solvers and theorem provers for quantified linear arithmetic benchmarks.

1.1 Preliminaries

We consider formulas in multi-sorted first-order logic. A *signature* Σ consists of a countable set of sort symbols and a set of function symbols. Given a signature Σ, well-sorted terms, atoms, literals, and formulas are defined as usual, and referred to respectively as Σ*-terms*. We denote by $FV(t)$ the set of free variables occurring in the term t, and extend this notion to formulas. A Σ-term or formula is *ground* if it has no free variables. We will consider term tuples $(t_1, \ldots, t_n)$, and denote them by letters in bold font, e.g. $\mathbf{t}$. A term written $t[\mathbf{k}]$ denotes a term whose free variables are in the tuple $\mathbf{k}$. A formula is *closed* if it has no free variables.

A Σ*-interpretation* $\mathcal{I}$ maps

- each set sort symbol $\tau \in \Sigma$ to a non-empty set $\tau^{\mathcal{I}}$, the *domain* of τ in $\mathcal{I}$,
- each function $f \in \Sigma$ of sort $\tau_1 \times \cdots \times \tau_n \to \tau$ to a total function $f^{\mathcal{I}}$ of sort $\tau_1^{\mathcal{I}} \times \cdots \times \tau_n^{\mathcal{I}} \to \tau^{\mathcal{I}}$ where $n > 0$, and to an element of $\tau^{\mathcal{I}}$ when $n = 0$, and
- each variable x of sort τ to an element of $\tau^{\mathcal{I}}$.

We write $t^{\mathcal{I}}$ to denote the interpretation of t in $\mathcal{I}$, defined inductively as usual. A satisfiability relation between Σ-interpretations and Σ-formulas, written $\mathcal{I} \models \varphi$, is also defined inductively as usual. In particular, we assume that $\mathcal{I} \models \neg\varphi$ if and only if it is not the case that $\mathcal{I} \models \varphi$. We say that $\mathcal{I}$ is *a model of* φ if $\mathcal{I}$ satisfies φ. Formulas φ_1 and φ_2 are *equivalent (up to* $\mathbf{k}$) if they are satisfied by the same set of models (when restricted to the interpretation of variables $\mathbf{k}$).

A *theory* is a pair $T = (\Sigma, \mathfrak{I})$ where Σ is a signature and $\mathfrak{I}$ is a non-empty set of Σ-interpretations, the *models* of T. We assume Σ contains the equality predicate, which we denote by $\approx$. Let $[\![\varphi]\!]_T$ denote the set of T-models of φ. Observe that $[\![\neg\varphi]\!]_T = \mathfrak{I} \setminus [\![\varphi]\!]_T$. A Σ-formula $\varphi[\mathbf{x}]$ is T*-satisfiable* if it is satisfied by some interpretation in $\mathfrak{I}$ (i.e. $[\![\varphi]\!]_T \neq \emptyset$). Dually, a Σ-formula $\varphi[\mathbf{x}]$ is T*-unsatisfiable* if it is satisfied by no interpretation in $\mathfrak{I}$ (i.e. $[\![\varphi]\!]_T = \emptyset$). A formula φ is T*-valid* if every model of T is a model of φ (i.e., $[\![\varphi]\!]_T = \mathfrak{I}$).

A set Γ of formulas T*-entails* a Σ-formula φ, written $\Gamma \models_T \varphi$, if every model of T that satisfies all formulas in Γ satisfies φ as well. A set of literals M *propositionally entails* a formula φ, written $M \models_p \varphi$, if M entails φ when considering all atomic formulas in $M \cup \varphi$ as propositional variables; such entailment is that of propositional logic and is independent of the theory.

We write RA (resp. IA) to denote the theory of real (resp. integer) arithmetic. Its signature consists of the sort Real (resp. Int), the binary predicate symbols $>$ and $<$, functions $+$ and $\cdot$ denoting addition and multiplication, and the constants of its sort interpreted as usual. We write $t \leq s$ as shorthand for $\neg(t > s)$, and $t \geq s$ as shorthand for $\neg(t < s)$. We write LRA (resp. LIA) to denote the language of linear real (resp. integer) arithmetic formulas, that is, whose literals are of the form $(\neg)(c_1 \cdot x_1 + \cdots + c_n \cdot x_n \bowtie c)$ where $c_1, \ldots, c_n, c$ and $x_1, \ldots, x_n$ are non-zero constants and distinct variables of sort Real (resp. Int) respectively, and $\bowtie$ is one of $>$, $<$, or $\approx$. For each literal of this form, there exists an equivalent literal that is in *solved form with respect to* x_i for each $i = 1, \ldots, n$. That is, an LRA-literal is in solved form with respect to x if it is of the form $(\neg)(x \bowtie t)$, where $x \notin FV(t)$. Similarly, an LIA-literal is in solved form with respect to x if it is of the form $(\neg)(c \cdot x \bowtie t)$, where $x \notin FV(t)$ and c is an integer constant greater than zero. For integer constants c_1 and c_2 and non-zero constant c, we write $c_1 \equiv_c c_2$ to denote that c_1 and c_2 are congruent modulo c, that is $(c_1 \;\mathsf{mod}\; c) = (c_2 \;\mathsf{mod}\; c)$, and we write $c \mid c_1$ if c divides c_1.

We write LIRA to denote the language of mixed linear real and integer arithmetic formulas, that is, linear arithmetic formulas where variables and constants may be of either real or integer sort.

2 Counterexample-guided quantifier instantiation

In this section, we assume a fixed theory T and a language $\mathfrak{L}$ that is closed under negation and such that the satisfiability of finite sets of $\mathfrak{L}$ formulas modulo T is decidable. We present a procedure for checking satisfiability of formulas in the language $\mathcal{Q}(\mathfrak{L}) = \{\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}] \mid \varphi[\mathbf{k}, \mathbf{x}] \in \mathfrak{L}\}$.

2.1 An instantiation procedure and its soundness

Figure 1 presents an instantiation-based procedure for determining the satisfiability of a T-formulas $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$, where $\varphi[\mathbf{k}, \mathbf{x}]$ belongs to $\mathfrak{L}$. The procedure introduces a tuple of distinct fresh variables $\mathbf{e}$ of the same sort as $\mathbf{x}$. It maintains a set of formulas Γ, initially empty, and terminates when either Γ or $\Gamma \cup \{\neg\varphi[\mathbf{k}, \mathbf{e}]\}$ is T-unsatisfiable. On each iteration, the procedure invokes the subprocedure $\mathcal{S}$ (over which the procedure is parameterized), which returns a tuple of terms $\mathbf{t}[\mathbf{k}]$ whose free variables are a subset of $\mathbf{k}$. We then add to Γ the formula $\varphi[\mathbf{k}, \mathbf{t}[\mathbf{k}]]$. We call $\mathcal{S}$ the *selection function* of $\mathcal{P}_{\mathcal{S}}$.

Definition 1 A selection function (for $\mathfrak{L}$) takes as arguments an interpretation $\mathcal{I}$, a set of formulas Γ, and a formula $\neg\varphi[\mathbf{k}, \mathbf{e}]$ in $\mathfrak{L}$, and a tuple of variables $\mathbf{e}$, where $\mathcal{I} \models \Gamma \cup \neg\varphi[\mathbf{k}, \mathbf{e}]$. It returns a tuple of terms $\mathbf{t}[\mathbf{k}]$ such that $\varphi[\mathbf{k}, \mathbf{t}[\mathbf{k}]]$ is also in $\mathfrak{L}$.

The intuition of the algorithm in Fig. 1 is to find a subset of the instances of $\forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$ that are either (a) unsatisfiable, and are thus sufficient for showing that $\forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$ is unsatisfiable, or (b) satisfiable and entail $\forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$. The algorithm recognizes the latter case by checking the satisfiability of $\Gamma \cup \neg\varphi[\mathbf{k}, \mathbf{e}]$ on each iteration of its main loop. In either case, the algorithm may terminate before enumerating all instances of $\forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$.

The procedure above is agnostic to the theory T. We remark that similar instantiation-based procedures have been developed in recent SMT solvers [19,50]. In this paper, we will focus on instantiation-based procedures for linear arithmetic, giving technical comparisons to existing approaches when applicable.

We first show that the procedure always returns correct results, regardless of the behavior of the selection function, leaving the termination question for the next subsection. We first prove that when the procedure terminates, the input to the procedure is equivalent to Γ. This means that the procedure $\mathcal{P}_{\mathcal{S}}$ can perform quantifier elimination by means of tracking the contents of Γ.

Lemma 1 *If $\mathcal{P}_{\mathcal{S}}$ terminates when $\Gamma = \{\varphi[\mathbf{k}, \mathbf{t}_1], \dots, \varphi[\mathbf{k}, \mathbf{t}_n]\}$, then $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$ is equivalent to $\exists \mathbf{k}\, \varphi[\mathbf{k}, \mathbf{t}_1] \wedge \cdots \wedge \varphi[\mathbf{k}, \mathbf{t}_n]$.*

```
$\mathcal{P}_{\mathcal{S}}(\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}])$:
    Let $\Gamma := \emptyset$ and $\mathbf{e}$ be a tuple of fresh variables of the same type as $\mathbf{x}$.
    Repeat
        If $\Gamma$ is $T$-unsatisfiable, then return "unsat".
        If $\Gamma' = \Gamma \cup \{\neg\varphi[\mathbf{k}, \mathbf{e}]\}$ is $T$-unsatisfiable, then return "sat".
        Otherwise,
            Let $\mathcal{I}$ be a model of $T$ and $\Gamma'$ and let $\mathbf{t}[\mathbf{k}] = \mathcal{S}(\mathcal{I}, \Gamma, \neg\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e})$.
            $\Gamma := \Gamma \cup \{\varphi[\mathbf{k}, \mathbf{t}[\mathbf{k}]]\}$.
```

Fig. 1 An instantiation-based procedure $\mathcal{P}_{\mathcal{S}}$ for determining the T-satisfiability of $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$, parameterized by selection function $\mathcal{S}$

Proof First, clearly all models of $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$ also satisfy $\exists \mathbf{k}\, \varphi[\mathbf{k}, \mathbf{t}_1] \wedge \cdots \wedge \varphi[\mathbf{k}, \mathbf{t}_n]$, since each $\varphi[\mathbf{k}, \mathbf{t}_i]$ is a consequence of $\forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$.

To show the opposite direction of the equivalence, in the case that $\mathcal{P}_\mathcal{S}$ terminates with "unsat", we have that Γ is T-unsatisfiable, and thus $\exists \mathbf{k}\, \varphi[\mathbf{k}, \mathbf{t}_1] \wedge \cdots \wedge \varphi[\mathbf{k}, \mathbf{t}_n]$ is T-unsatisfiable. Thus, it is vacuously the case that all of its models satisfy $\forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$. In the case that $\mathcal{P}_\mathcal{S}$ terminates with "sat", we have that Γ is T-satisfiable and $\Gamma' = \Gamma \cup \{\neg\varphi[\mathbf{k}, \mathbf{e}]\}$ is T-unsatisfiable. Since the only formulas added to Γ are of the form $\varphi[\mathbf{k}, \mathbf{t}[\mathbf{k}]]$, the variables $\mathbf{e}$ do not occur in Γ, and thus $\Gamma \cup \{\exists \mathbf{x}\, \neg\varphi[\mathbf{k}, \mathbf{x}]\}$ is T-unsatisfiable as well. Let $\mathcal{I}$ be a model of Γ. Since $\mathcal{I}$ is not a model of $\Gamma \cup \{\exists \mathbf{x}\, \neg\varphi[\mathbf{k}, \mathbf{x}]\}$, it must be the case that $\mathcal{I} \not\models \exists \mathbf{x}\, \neg\varphi[\mathbf{k}, \mathbf{x}]$, and hence $\mathcal{I}$ is a model for $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$. Thus, in this case all models of $\exists \mathbf{k}\, \varphi[\mathbf{k}, \mathbf{t}_1] \wedge \cdots \wedge \varphi[\mathbf{k}, \mathbf{t}_n]$ also satisfy $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$. Thus, the lemma holds. □

Corollary 1 *If $\mathcal{P}_\mathcal{S}$ terminates with "unsat", then $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$ is T-unsatisfiable.*

Proof $\mathcal{P}_\mathcal{S}$ terminates with "unsat" only if Γ is T-unsatisfiable. Thus by Lemma 1, we have that $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$ is T-unsatisfiable as well. □

Corollary 2 *If $\mathcal{P}_\mathcal{S}$ terminates with "sat", then $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$ is T-satisfiable.*

Proof $\mathcal{P}_\mathcal{S}$ terminates with "sat" only if Γ is T-satisfiable. Thus by Lemma 1, we have that $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$ is T-satisfiable as well. □

2.2 Termination of the instantiation procedure

The following properties of selection functions are of interest for showing the procedure $\mathcal{P}_\mathcal{S}$ terminates.

Definition 2 (*Finite*) A selection function $\mathcal{S}$ is *finite* for $\varphi[\mathbf{k}, \mathbf{e}]$ if there exists a finite set $\mathcal{S}^*(\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e})$ such that $\mathcal{S}(\mathcal{I}, \Gamma, \neg\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e}) \in \mathcal{S}^*(\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e})$ for all $\mathcal{I}, \Gamma$.

Definition 3 (*Monotonic*) A selection function $\mathcal{S}$ is *monotonic* for $\varphi[\mathbf{k}, \mathbf{e}]$ if whenever $\Gamma \models \varphi[\mathbf{k}, \mathbf{t}]$, we have that $\mathcal{S}(\mathcal{I}, \Gamma, \neg\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e}) \neq \mathbf{t}$.

Observe that, if $\mathcal{S}$ is a monotonic selection function, then for any finite list of terms $\mathbf{t}_1, \ldots \mathbf{t}_n$ we have $\mathcal{S}(\mathcal{I}, \{\varphi[\mathbf{k}, \mathbf{t_1}], \ldots, \varphi[\mathbf{k}, \mathbf{t_n}]\}, \neg\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e}) \notin \{\mathbf{t}_1, \ldots, \mathbf{t}_n\}$.

Definition 4 (*Model-Preserving*) A selection function $\mathcal{S}$ is *model-preserving* for $\varphi[\mathbf{k}, \mathbf{e}]$ if whenever $\mathcal{S}(\mathcal{I}, \Gamma, \neg\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e}) = \mathbf{t}$, we have that $\mathcal{I} \models \neg\varphi[\mathbf{k}, \mathbf{t}]$.

Lemma 2 *A selection function that is model-preserving for $\varphi[\mathbf{k}, \mathbf{e}]$ is also monotonic for $\varphi[\mathbf{k}, \mathbf{e}]$.*

Proof Assume that $\mathcal{S}$ is model-preserving for $\varphi[\mathbf{k}, \mathbf{e}]$ and that $\mathcal{S}(\mathcal{I}, \Gamma, \neg\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e}) = \mathbf{s}$. By definition of model-preserving, we have that $\mathcal{I} \models \neg\varphi[\mathbf{k}, \mathbf{s}]$. Thus, for each $\mathbf{t}$ such that $\mathcal{I} \models \varphi[\mathbf{k}, \mathbf{t}]$, we have that $\mathbf{s} \neq \mathbf{t}$. Thus, $\mathcal{S}$ is monotonic for $\varphi[\mathbf{k}, \mathbf{e}]$. □

Theorem 1 *If $\mathcal{S}$ is finite and monotonic for $\varphi[\mathbf{k}, \mathbf{e}]$ in $\mathfrak{L}$, then $\mathcal{P}_\mathcal{S}$ is a (terminating) decision procedure for the T-satisfiability of $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$.*

Proof Given a monotonic and finite $\mathcal{S}$, the procedure $\mathcal{P}_\mathcal{S}$ can only execute a finite number of iterations. Assuming a decision procedure for determining the T-satisfiability of T-formulas in $\mathfrak{L}$, $\mathcal{P}_\mathcal{S}(\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}])$ must terminate. By Corollaries 1 and 2, $\mathcal{P}_\mathcal{S}$ is a decision procedure for the T-satisfiability of $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$. □

By this result, we obtain a sound and complete instantiation strategy from Fig. 1 by virtue of constructing a selection function that is finite and monotonic for formulas residing in a language $\mathfrak{L}$ of interest. It is important to note that the property of being both finite and monotonic is a sufficient condition of a selection function for proving the termination of an instantiation-based procedure according to the algorithm in Fig. 1, although it is not the only sufficient condition for doing so.

In this paper we will construct selection functions $\mathcal{S}$ that are finite and monotonic for all $\varphi[\mathbf{k}, \mathbf{x}]$ in quantifier-free linear real and integer arithmetic.

3 Instantiation for LRA-formulas

Consider the case where $\mathbf{k}$ and $\mathbf{x}$ are vectors of Real variables and $\mathfrak{L}$ is the class of quantifier-free LRA-formulas $\varphi[\mathbf{k}, \mathbf{x}]$. For simplicity of the presentation, we assume that equalities are eliminated from φ by the transformation:

$$t \approx 0 \rightsquigarrow 0 \leq t \wedge 0 \geq t$$

As a result of Theorem 1, to devise a sound and complete instantiation-based procedure for deciding the satisfiability of $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$, it suffices to devise a finite and monotonic selection function for LRA, which we describe in the following.

Figure 2 gives a selection function $\mathcal{S}_{\text{LRA}}$ for LRA, which takes an interpretation $\mathcal{I}$, a set of formulas Γ, the formula $\neg\varphi[\mathbf{k}, \mathbf{e}]$, and the tuple of variables $\mathbf{e}$. It invokes the recursive procedure S_R which constructs a term corresponding to each variable in $\mathbf{e}$. Analogous to existing approaches for linear quantifier elimination [36,43], we assume that the signature of linear real arithmetic contains non-standard terms for symbolically representing substitutions (often referred to as *virtual terms*). In particular, we consider a signature that contains a free distinguished constant δ of sort Real, representing an infinitesimal positive value. Any

```
$\mathcal{S}_{\text{LRA}}(\mathcal{I}, \Gamma, \neg\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e})$:
    Return $S_R(\mathcal{I}, \neg\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e}, ())$
$S_R(\mathcal{I}, \psi, (\mathsf{e}_i, \ldots, \mathsf{e}_n), \mathbf{t})$:
    If $i > n$, return $\mathbf{t}$
    Otherwise, let $t_i = S_{R0}(\mathcal{I}, \psi, \mathsf{e}_i)$, $\sigma = \{\mathsf{e}_i \mapsto t_i\}$
    Return $S_R(\mathcal{I}, \psi\sigma, (\mathsf{e}_{i+1}, \ldots, \mathsf{e}_n), (\mathbf{t}\sigma, t_i))$
$S_{R0}(\mathcal{I}, \psi, \mathsf{e})$:
    Let $M = M_\ell \cup M_u \cup M_c$ be such that:
      – $\mathcal{I} \models M$ and $M \models_p \psi$,
      – $M_\ell \Leftrightarrow \{\mathsf{e} \succ \ell_1, \ldots, \mathsf{e} \succ \ell_n\}$,
      – $M_u \Leftrightarrow \{\mathsf{e} \prec u_1, \ldots, \mathsf{e} \prec u_m\}$, and
      – $\mathsf{e} \notin FV(\ell_1, \ldots, \ell_n) \cup FV(u_1, \ldots, u_m) \cup FV(M_c)$.
    Return one of $\begin{cases} \ell_i + \delta^\ell_i & n > 0, \max\{(\ell_1 + \delta^\ell_1)^\mathcal{I}, \ldots, (\ell_n + \delta^\ell_n)^\mathcal{I}\} = (\ell_i + \delta^\ell_i)^\mathcal{I} \\ u_j - \delta^u_j & m > 0, \min\{(u_1 - \delta^u_1)^\mathcal{I}, \ldots, (u_m - \delta^u_m)^\mathcal{I}\} = (u_j - \delta^u_j)^\mathcal{I} \\ 0 & n = 0 \text{ and } m = 0 \end{cases}$
```

Fig. 2 A selection function $\mathcal{S}_{\text{LRA}}$ for linear real arithmetic LRA. Each $\prec$ is either $<$ or $\leq$; δ^ℓ_i is δ if the ith lower bound for e is strict, and 0 otherwise. Similarly, each $\succ$ is either $>$ or $\geq$; δ^u_j is δ if the jth upper bound for e is strict, and 0 otherwise

quantifier-free constraint containing δ is equivalent to one that does not by the transformations:

$$\delta < t \rightsquigarrow 0 < t \quad \text{and} \quad \delta > t \rightsquigarrow 0 \geq t \quad \text{where } \delta \notin FV(t).$$

Thus we may assume without loss of generality that $\neg\varphi[\mathbf{k}, \mathbf{e}]$ contains no occurrence of δ.

For each variable e_i from $\mathbf{e}$, the procedure S_R invokes the (non-deterministic) subprocedure S_{R0}, which chooses a term corresponding to e_i based on a set of literals M over the atoms of ψ which propositionally entail ψ and are satisfied by $\mathcal{I}$, which we call a *propositionally satisfying assignment* for ψ. We partition M into three sets M_ℓ, M_u and M_c, where M_ℓ contains literals that correspond to lower bounds for e, M_u contains literals that correspond to upper bounds for e, and M_c contains the remaining literals. The sets M_ℓ and M_u are equivalent to sets of literals that are in solved form with respect to e. When M_ℓ contains at least one literal, we may return the lower bound whose value is maximal according to $\mathcal{I}$, and similarly for M_u. If both M_ℓ and M_u are empty, we return the term 0. When S_{R0} returns the term t_i, we apply the substitution $\{\mathsf{e}_i \mapsto t_i\}$ to ψ and $\mathbf{t}$, and append t_i to $\mathbf{t}$. Terms returned by S_{R0} may involve the constant δ. We define a satisfiability relation between models and formulas involving δ, as well as the max and min function for terms involving δ in the obvious way, such that $(t_1 + c_1 \cdot \delta)^{\mathcal{I}} > (t_2 + c_2 \cdot \delta)^{\mathcal{I}}$ if either $t_1^{\mathcal{I}} > t_2^{\mathcal{I}}$ or both $t_1^{\mathcal{I}} = t_2^{\mathcal{I}}$ and $c_1 > c_2$.

Overall, $\mathcal{S}_{\text{LRA}}$ returns a tuple of terms $\mathbf{t}$, after which we add $\varphi[\mathbf{k}, \mathbf{t}]$ to Γ in Fig. 1.

Lemma 3 $\mathcal{S}_{\text{LRA}}$ *is finite for* $\varphi[\mathbf{k}, \mathbf{e}]$.

Proof We first show only a finite number of terms can be returned by $S_{R0}(\mathcal{I}, (\neg\varphi[\mathbf{k}, \mathbf{e}])\sigma, \mathsf{e})$ for any $\mathcal{I}, \sigma, \mathsf{e}$. Let A be the set of atoms occurring in $\varphi[\mathbf{k}, \mathbf{e}]\sigma$. The literals in satisfying assignments of $(\neg\varphi[\mathbf{k}, \mathbf{e}])\sigma$ are over these atoms. Let $\{\mathsf{e} > t_1, \ldots, \mathsf{e} > t_n, \mathsf{e} < s_1, \ldots, \mathsf{e} < s_m\}$ be the set of atoms that are in solved form with respect to e that are equivalent to the atoms of A containing e, where $\mathsf{e} \notin FV(t_1, \ldots, t_n, s_1, \ldots, s_m)$. The terms returned by $S_{R0}(\mathcal{I}, (\neg\varphi[\mathbf{k}, \mathbf{e}])\sigma, \mathsf{e})$ are in the set:

$$\{0, t_1(+\delta), \ldots, t_n(+\delta), s_1(-\delta), \ldots, s_m(-\delta)\}$$

Notice that literals over the atoms in A may occur either with positive or negative polarity in M. Thus for each literal $\mathsf{e} < t_i$ for $i = 1, \ldots, n$, S_{R0} may either return $t_i + \delta$ when considering $(\mathsf{e} > t_i)$ in M as a lower bound for e, or t_i when considering $\neg(\mathsf{e} > t_i)$ in M, which is equivalent to $(\mathsf{e} \leq t_i)$, as an upper bound for e. Similarly we consider two cases for each literal $\mathsf{e} > s_i$ for $i = 1, \ldots, m$. Since there are only a finite number of recursive calls to S_R within $\mathcal{S}_{\text{LRA}}$, and each call appends only a finite number of possible terms to $\mathbf{t}$, the set of possible return values of $\mathcal{S}_{\text{LRA}}$ is finite, and thus it is finite for $\varphi[\mathbf{k}, \mathbf{e}]$. □

Lemma 4 *If* $\mathcal{I}$ *is a model for* LRA *and for the quantifier-free formula* ψ*, then* $\mathcal{I}$ *is also a model for* $\psi\{\mathsf{e} \mapsto S_{R0}(\mathcal{I}, \psi, \mathsf{e})\}$.

Proof Let M be a set of literals of the form described in the definition of S_{R0} for $\mathcal{I}$, ψ and e. Consider the case where $S_{R0}(\mathcal{I}, \psi, \mathsf{e}) = \ell_i + \delta_i^\ell$ for some i, where $n > 0$. We show that $\mathcal{I}$ satisfies $M\{\mathsf{e} \mapsto \ell_i + \delta_i^\ell\}$. First, since $\mathsf{max}\{(\ell_1 + \delta_1^\ell)^{\mathcal{I}}, \ldots, (\ell_n + \delta_n^\ell)^{\mathcal{I}}\} = (\ell_i + \delta_i^\ell)^{\mathcal{I}}$, we know that $\mathcal{I}$ satisfies $M_\ell\{\mathsf{e} \mapsto \ell_i + \delta_i^\ell\}$. In the case that the bound on e we consider is strict, that is, $\mathsf{e} > \ell_i \in M_\ell$, then δ_i^ℓ is δ, and $\ell_i^{\mathcal{I}} < u_j^{\mathcal{I}}$ for all $j \in \{1, \ldots, m\}$. Thus, $\mathcal{I}$ satisfies $(\ell_i + \delta \prec u_j) = (\mathsf{e} \prec u_j)\{\mathsf{e} \mapsto \ell_i + \delta\}$. In the case that the bound on e we consider is non-strict, that is, if $\mathsf{e} \geq \ell_i \in M_\ell$, then δ_i^ℓ is 0, and $\ell_i^{\mathcal{I}} \leq u_j^{\mathcal{I}}$ for all $j \in \{1, \ldots, m\}$. Thus, $\mathcal{I}$ satisfies $(\ell_i \prec u_j) = (\mathsf{e} \prec u_j)\{\mathsf{e} \mapsto \ell_i\}$. In either case, we have that $\mathcal{I}$ satisfies each literal in $M_u\{\mathsf{e} \mapsto \ell_i + \delta_i^\ell\}$. Finally, $\mathcal{I}$ clearly satisfies $M_c\{\mathsf{e} \mapsto \ell_i + \delta_i^\ell\} = M_c$. The case when $m > 0$ is symmetric to the case when $n > 0$. In the case where $n = 0$ and $m = 0$, we have that ψ does not contain e, and $\mathcal{I}$ satisfies $M\{\mathsf{e} \mapsto 0\}$. In each case, $\mathcal{I}$ satisfies $M\{\mathsf{e} \mapsto S_{R0}(\mathcal{I}, \psi, \mathsf{e})\}$, which entails $\psi\{\mathsf{e} \mapsto S_{R0}(\mathcal{I}, \psi, \mathsf{e})\}$, and thus the lemma holds. □

Lemma 5 $\mathcal{S}_{\text{LRA}}$ *is model-preserving for* $\varphi[\mathbf{k}, \mathbf{e}]$.

Proof Assume $\mathcal{S}_{\text{LRA}}(\mathcal{I}, \Gamma, \neg\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e})$ returns $\mathbf{t}$. By definition of selection function, $\mathcal{I}$ and $\neg\varphi[\mathbf{k}, \mathbf{e}]$ are such that $\mathcal{I} \models \neg\varphi[\mathbf{k}, \mathbf{e}]$. By repeated applications of Lemma 4, we have that $\mathcal{I}$ satisfies all inputs ψ to S_R. When S_R terminates, ψ is $\neg\varphi[\mathbf{k}, \mathbf{t}]$, and thus $\mathcal{I} \models \neg\varphi[\mathbf{k}, \mathbf{t}]$. □

Theorem 2 $\mathcal{P}_{\mathcal{S}_{\text{LRA}}}$ *is a sound and complete procedure for determining the* LRA*-satisfiability of* $\exists\mathbf{k}\,\forall\mathbf{x}\,\varphi[\mathbf{k}, \mathbf{x}]$.

Proof By Theorem 1 and Lemma 2 of our framework as well as LRA-specific Lemma 3 and Lemma 5. □

We illustrate the procedure through examples. S_{R0} is non-deterministic; we choose instantiations only based on the lower bounds M_ℓ found in the procedure S_{R0}, though the procedure is free to choose its instantiations based on the upper bounds M_u as well. We underline the literal in M_ℓ corresponding to the bound whose value is maximal in $\mathcal{I}$. Γ is initially empty and on each iteration Γ' is the union of Γ and the *Skolemized negation* $\neg\varphi[\mathbf{e}]$ of the input formula $\forall\mathbf{x}\,\varphi[\mathbf{x}]$, where $\mathbf{e}$ are fresh variables of the same sort as $\mathbf{x}$. Each round of $\mathcal{S}_{\text{LRA}}$ computes a tuple $\mathbf{t}[\mathbf{k}]$, which is used to instantiate our quantified formula in Fig. 1. The last column shows the corresponding instance of the quantified formula after simplification, including the elimination of δ.

Example 1 Consider the formula $\forall x\,(x \le a \vee x \le b)$. Let e be a fresh Skolem variable of the same sort (Real) as x. The Skolemized negation of this formula is $\neg(e \le a \vee e \le b)$, which simplifies to $(e > a \wedge e > b)$. The iterations of the loop of $\mathcal{P}_{\mathcal{S}_{\text{LRA}}}$ are summarized in the following table, where Γ is initially empty, and Γ' is obtained by adding $(e > a \wedge e > b)$ to Γ on each iteration.

			$S_{R0}(\mathcal{I}, \psi, \mathsf{e})$				
#	Γ	Γ'	e	M_ℓ	Return	$\mathbf{t}[\mathbf{k}]$	Add to Γ
1	sat	sat	e	$\{\underline{e > a}, e > b\}$	$a + \delta$	$(a + \delta)$	$a < b$
2	sat	sat	e	$\{e > a, \underline{e > b}\}$	$b + \delta$	$(b + \delta)$	$b < a$
3	unsat						

On the first iteration, Γ and $\Gamma' = \{e > a \wedge e > b\}$ are satisfiable by a model, call it $\mathcal{I}_1$, where assume that $e^{\mathcal{I}_1} = 2, a^{\mathcal{I}_1} = 1, b^{\mathcal{I}_1} = 0$. We call S_{R0} with inputs $\mathcal{I}_1$, $(e > a \wedge e > b)$, and e. A propositionally satisfying assignment M for $(e > a \wedge e > b)$ includes both these literals, and the set of lower bounds M_ℓ for e on this iteration is $\{e > a, e > b\}$. Since $a^{\mathcal{I}_1} > b^{\mathcal{I}_1}$, the procedure S_{R0} finds that the literal $e > a$ gives the maximal lower bound for e, and hence it returns the term $a + \delta$. The disjuncts of the instance $a + \delta \le a \vee a + \delta \le b$ added to Γ on the first iteration simplify to $\perp$ and $a < b$ respectively. On the second iteration, Γ and $\Gamma' = \{e > a \wedge e > b, a < b\}$ are still satisfiable with a model, call it $\mathcal{I}_2$, where assume that $e^{\mathcal{I}_2} = 2, a^{\mathcal{I}_2} = 0, b^{\mathcal{I}_2} = 1$. Since now $b^{\mathcal{I}_2} > a^{\mathcal{I}_2}$, we have that b is now the maximal lower bound for e. Hence, S_{R0} returns $b+\delta$ on the second iteration, and we add the formula $b < a$ to Γ. On the third iteration, Γ contains both $a < b$ and $b < a$ and hence is LRA-unsatisfiable. Overall, this run shows $\exists ab\, \forall x\, (a \geq x \vee x \geq b)$ is LRA-unsatisfiable. □

Example 2 To demonstrate how multiple universally quantified variables are handled, consider the formula $\forall xy\, (x+y < a \vee x-y < b)$ whose Skolemized negation after simplification is $e_1 + e_2 \geq a \wedge e_1 - e_2 \geq b$. A possible run of $\mathcal{P}_{\mathcal{S}_{\text{LRA}}}$ is as follows.

#	Γ	Γ'	$S_{R0}(\mathcal{I}, \psi, \mathsf{e})$		Return	$\mathbf{t}[\mathbf{k}]$	Add to Γ
			e	M_ℓ			
1	sat	sat	e_1	$\{e_1 \geq a - e_2, \underline{e_1 \geq b + e_2}\}$	$b + e_2$		
			e_2	$\{\underline{e_2 \geq \frac{a-b}{2}}\}$	$\frac{a-b}{2}$	$(\frac{a+b}{2}, \frac{a-b}{2})$	$\perp$
2	unsat						

On the first iteration, since Γ and Γ' are satisfiable say with model $\mathcal{I}_1$, we first call S_{R0} on $\mathcal{I}_1$, $(e_1 + e_2 \geq a \wedge e_1 - e_2 \geq b)$, and e_1. The set M_ℓ contains inequalities that are in solved form with respect to e_1 that correspond to lower bounds for e_1 in $(e_1 + e_2 \geq a \wedge e_1 - e_2 \geq b)$, which are $\{e_1 \geq a - e_2 \wedge e_1 \geq b + e_2\}$. Assuming $b + e_2$ is the maximal lower bound for e_1, S_{R0} returns $b + e_2$. In the procedure S_R, we then apply the substitution $\{e_1 \mapsto b + e_2\}$ to $(e_1 + e_2 \geq a \wedge e_1 - e_2 \geq b)$, giving us $(b + e_2 + e_2 \geq a \wedge b + e_2 - e_2 \geq b)$, which simplifies to $e_2 \geq \frac{a-b}{2}$. Calling S_{R0} again with input $\mathcal{I}_1$, this formula, and e_2, we have that M_ℓ now is the set $\{e_2 \geq \frac{a-b}{2}\}$, and thus S_{R0} returns $\frac{a-b}{2}$ for e_2. We apply the substitution $\{e_2 \mapsto \frac{a-b}{2}\}$ to the term $b + e_2$ we computed for e_1, giving us $b + \frac{a-b}{2}$, which simplifies to $\frac{a+b}{2}$. Overall, $\mathcal{S}_{\text{LRA}}$ returns the tuple of terms $(\frac{a+b}{2}, \frac{a-b}{2})$. Applying the substitution $\{x \mapsto \frac{a+b}{2}, y \mapsto \frac{a-b}{2}\}$ to our input formula results in the formula $(\frac{a+b}{2} + \frac{a-b}{2} < a \vee \frac{a+b}{2} - \frac{a-b}{2} < b)$, which simplifies to $\perp$. This run shows $\exists ab\, \forall xy\, (x + y < a \vee x - y < b)$ is LRA-unsatisfiable. □

Example 3 To demonstrate how quantified formulas with Boolean structure are handled, consider the formula $\forall x\, ((a < x \wedge x < b) \vee x < a + b)$ whose Skolemized negation is $(a \geq e \vee e \geq b) \wedge e \geq a + b$. A possible run of $\mathcal{P}_{\mathcal{S}_{\text{LRA}}}$ is as follows.

#	Γ	Γ'	$S_{R0}(\mathcal{I}, \psi, \mathsf{e})$		Return	$\mathbf{t}[\mathbf{k}]$	Add to Γ
			e	M_ℓ			
1	sat	sat	e	$\{\underline{e \geq a + b}\}$	$a + b$	$(a + b)$	$0 < b \wedge a < 0$
2	sat	sat	e	$\{\underline{e \geq b}, e \geq a + b\}$	b	(b)	$0 < a$
3	unsat						

On the first iteration, we have that Γ and $\Gamma' = \{(a \geq e \vee e \geq b) \wedge e \geq a + b\}$ are satisfiable with a model, call it $\mathcal{I}_1$. In the above run, we assume that $\mathcal{I}_1$ satisfies $a \geq e$ but not $e \geq b$, and hence $e \geq b$ is not included as a lower bound in M_ℓ. We return the instance for $\{x \mapsto a + b\}$, which simplifies to $0 < b \wedge a < 0$. On the second iteration, the model for Γ', call it $\mathcal{I}_2$, must now satisfy $0 < b \wedge a < 0$, and hence $b^{\mathcal{I}_2} > (a + b)^{\mathcal{I}_2} > a^{\mathcal{I}_2}$. Since $\mathcal{I}_2$ satisfies $e \geq a + b$, it cannot satisfy $a \geq e$, and hence it must satisfy $e \geq b$. Thus, on this iteration, M_ℓ contains $e \geq b$ and $e \geq a + b$. Moreover since $b^{\mathcal{I}_2} > (a + b)^{\mathcal{I}_2}$, we know that b must be the maximal lower bound for e, and hence we return an instance based on $\{x \mapsto b\}$, which simplifies to $0 < a$, after which we find that Γ is unsatisfiable. This run shows $\exists ab\, \forall x\, ((a < x \wedge x < b) \vee x < a + b)$ is LRA-unsatisfiable. □

Example 4 To demonstrate a case where a variable has no bounds, consider the formula $\forall xy\,(x \leq y)$, whose Skolemized negation is $e_1 > e_2$. A possible run of $\mathcal{P}_{\mathcal{S}_{\text{LRA}}}$ on this input is as follows.

#	Γ	Γ'	$S_{R0}(\mathcal{I}, \psi, \mathsf{e})$		Return	$\mathbf{t}[\mathbf{k}]$	Add to Γ
			e	M_ℓ			
1	sat	sat	e_1	$\{\underline{e_1 > e_2}\}$	$e_2 + \delta$		
			e_2	$\emptyset$	0	$(\delta, 0)$	$\bot$
2	unsat						

On the first iteration, after choosing $e_2 + \delta$ for e_1, we apply the substitution $\{e_1 \mapsto e_2 + \delta\}$ to Γ', giving us the set containing $e_2 + \delta > e_2$, which simplifies to $\top$. Thus when using S_{R0} to choose a term for e_2, we have that M_ℓ contains neither an upper nor a lower bound for e_2, and hence we choose to return the value 0. The instance of our input formula corresponding to the substitution $\{x \mapsto \delta, y \mapsto 0\}$ simplifies to $\bot$. This run shows $\forall xy\,(x \leq y)$ is LRA-unsatisfiable. □

Example 5 To demonstrate a non-trivial case using the infinitesimal δ, consider the formula $\forall xy\,(x \leq 0 \vee y - 2 \cdot x \leq 0)$ whose Skolemized negation is $e_1 > 0 \wedge e_2 - 2 \cdot e_1 > 0$. A possible run of $\mathcal{P}_{\mathcal{S}_{\text{LRA}}}$ on this input is as follows.

#	Γ	Γ'	$S_{R0}(\mathcal{I}, \psi, \mathsf{e})$		Return	$\mathbf{t}[\mathbf{k}]$	Add to Γ
			e	M_ℓ			
1	sat	sat	e_1	$\{\underline{e_1 > 0}\}$	δ		
			e_2	$\{\underline{e_2 > 2 \cdot \delta}\}$	$3 \cdot \delta$	$(\delta, 3 \cdot \delta)$	$\bot$
2	unsat						

The instance corresponding to the substition $\{x \mapsto \delta, y \mapsto 3 \cdot \delta\}$ is $(\delta \leq 0 \vee 3 \cdot \delta - 2 \cdot \delta \leq 0)$, which simplifies to $(\delta \leq 0 \vee \delta \leq 0)$, which after eliminating δ simplifies to $\bot$. This run shows $\forall xy\,(x \leq 0 \vee y - 2 \cdot x \leq 0)$ is LRA-unsatisfiable. □

The procedure $\mathcal{P}_{\mathcal{S}_{\text{LRA}}}$, which is an instance of the procedure in Fig. 1, can be understood as lazily enumerating the disjuncts of the Loos–Weispfenning method for quantifier elimination over linear real arithmetic [36], with minor differences which we discuss in the next section. In this way, our approach is similar to the projection-based procedures described in [13,32]. These approaches compute implicants of quantified formulas, while our approach instead computes a term which is in turn used for instantiation. For our purposes, computing a term instead of an implicant has several advantages. In particular, it allows the instantiation-based procedure to be used as a subprocedure for synthesis, which we describe in Sect. 7, and enables a uniform combination of the approach with existing instantiation-based techniques for first-order logic [18,24,51].

3.1 Comparison to existing approaches

Recent approaches (including ours) for solving quantified linear arithmetic share similarities with one another. In particular, given an existentially quantified formula $\exists \mathbf{x}.\varphi$, based on some strategy, they enumerate (possibly lazily) a finite set of ground formulas that are entailed by this formula. We give a brief overview contrasting the technical details of existing approaches in this section.

We have mentioned that the approach in Fig. 2 involves the use of a free distinguished constant δ, representing an infinitesimal positive value. Other approaches also involve use of a free distinguished constant ∞, representing an arbitrarily large positive value. Like δ, this term can be eliminated from quantifier-free constraints, noting:

$$\infty < t \rightsquigarrow \bot \quad \text{and} \quad \infty > t \rightsquigarrow \top \quad \text{where } \infty \notin FV(t).$$

The two most widely known algorithms for quantifier elimination for linear real arithmetic are the method based on virtual term substitution by Loos and Weispfenning [36], and the method based on interior points by Ferrante and Rackoff [23]. In the context of our approach, two alternatives for the return value of S_{R0} (Figs. 3 and 4) closely approximate the effect of these methods.

Recent approaches are inspired by of one (or both) of these methods. The approaches described in [13,32] are closely based on the Loos–Weispfenning method, and the approach described in [19] is closely based on Ferrante–Rackoff method. The approach described in [43] examines a certified version of both approaches. The approach in Fig. 2 is inspired by the Loos–Weispfenning method, but does not use infinities.

Another possible return value is given in Fig. 5 that does not use any virtual terms. The approach in Fig. 5 may be advantageous when considering quantified formulas having nested quantification, where eliminating virtual terms from quantified constraints is not obvious.

4 Instantiation for LIA-formulas

We now turn our attention to the class of LIA-formulas $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$, where $\mathbf{x}$ and $\mathbf{k}$ are vectors of Int variables, and $\varphi[\mathbf{k}, \mathbf{x}]$ is quantifier-free. We again assume all equalities are

$$\text{Return one of } \begin{cases} \ell_i + \delta & n > 0 \\ u_j - \delta & m > 0 \\ \infty & m = 0 \\ -\infty & n = 0 \end{cases}, \text{ where } \begin{cases} \mathsf{max}\{\ell_1^{\mathcal{I}}, \ldots, \ell_n^{\mathcal{I}}\} = \ell_i^{\mathcal{I}} \text{ if } n > 0 \\ \mathsf{min}\{u_1^{\mathcal{I}}, \ldots, u_m^{\mathcal{I}}\} = u_j^{\mathcal{I}} \text{ if } m > 0. \end{cases}.$$

Fig. 3 An alternative return value for S_{R0} that is analogous to Loos and Weispfenning's method

$$\text{Return } \begin{cases} \frac{u_j+\ell_i}{2} & n > 0 \text{ and } m > 0 \\ \infty & m = 0 \\ -\infty & n = 0 \end{cases}, \text{ where } \begin{cases} \mathsf{max}\{\ell_1^{\mathcal{I}}, \ldots, \ell_n^{\mathcal{I}}\} = \ell_i^{\mathcal{I}} \text{ if } n > 0 \\ \mathsf{min}\{u_1^{\mathcal{I}}, \ldots, u_m^{\mathcal{I}}\} = u_j^{\mathcal{I}} \text{ if } m > 0. \end{cases}.$$

Fig. 4 An alternative return value for S_{R0} that is analogous to Ferrante and Rackoff's method

$$\text{Return } \begin{cases} \frac{u_j+\ell_i}{2} & n > 0 \text{ and } m > 0 \\ \ell_i + 1 & n > 0 \text{ and } m = 0 \\ u_j - 1 & n = 0 \text{ and } m > 0 \\ 0 & n = 0 \text{ and } m = 0 \end{cases}, \text{ where } \begin{cases} \mathsf{max}\{\ell_1^{\mathcal{I}}, \dots, \ell_n^{\mathcal{I}}\} = \ell_i^{\mathcal{I}} \text{ if } n > 0 \\ \mathsf{min}\{u_1^{\mathcal{I}}, \dots, u_m^{\mathcal{I}}\} = u_j^{\mathcal{I}} \text{ if } m > 0. \end{cases}.$$

Fig. 5 An alternative return value for S_{R0}

```
$\mathcal{S}_{\text{LIA}}(\mathcal{I}, \Gamma, \neg\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e})$:
    Return $S_I(\mathcal{I}, \neg\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e}, 1, (), ())$.
$S_I(\mathcal{I}, \psi, (\mathsf{e}_i, \dots, \mathsf{e}_n), \theta, \mathbf{t}, \mathbf{p})$:
    If $i > n$, return $\mathbf{t} \ \mathsf{div}^{\mathbf{p}} \ \theta$
    Otherwise, let $(c, t_i, p_i) = S_{I0}(\mathcal{I}, \psi, \mathsf{e}_i, \theta)$, $\sigma = \{c \cdot \mathsf{e}_i \mapsto t_i\}$
    Return $S_I(\mathcal{I}, \psi\sigma, (\mathsf{e}_{i+1}, \dots, \mathsf{e}_n), \theta \cdot c, ((c \cdot \mathbf{t})\sigma, \theta \cdot t_i), (\mathbf{p}, p_i))$
$S_{I0}(\mathcal{I}, \psi, \mathsf{e}, \theta)$:
    Let $M = M_\ell \cup M_u \cup M_c$ be such that:
      – $\mathcal{I} \models M$ and $M \models_p \psi$,
      – $M_\ell \Leftrightarrow \{c_1 \cdot \mathsf{e} \geq \ell_1, \dots, c_n \cdot \mathsf{e} \geq \ell_n\}$, $c_1 > 0, \dots, c_n > 0$,
      – $M_u \Leftrightarrow \{d_1 \cdot \mathsf{e} \leq u_1, \dots, d_m \cdot \mathsf{e} \leq u_m\}$, $d_1 > 0, \dots, d_m > 0$, and
      – $\mathsf{e} \notin FV(\ell_1, \dots, \ell_n) \cup FV(u_1, \dots, u_m) \cup FV(M_c)$.
    Return one of $\begin{cases} (c_i, \ell_i + \rho, +) & n > 0, \mathsf{max}\{(\frac{\ell_1}{c_1})^{\mathcal{I}}, \dots, (\frac{\ell_n}{c_n})^{\mathcal{I}}\} = (\frac{\ell_i}{c_i})^{\mathcal{I}}, \\ & \rho = (c_i \cdot \mathsf{e} - \ell_i)^{\mathcal{I}} \ \mathsf{mod} \ (\theta \cdot c_i) \\ (d_j, u_j - \rho, -) & m > 0, \mathsf{min}\{(\frac{u_1}{d_1})^{\mathcal{I}}, \dots, (\frac{u_m}{d_m})^{\mathcal{I}}\} = (\frac{u_j}{d_j})^{\mathcal{I}}, \\ & \rho = (u_j - d_j \cdot \mathsf{e})^{\mathcal{I}} \ \mathsf{mod} \ (\theta \cdot d_j) \\ (1, \rho, +) & n = 0, m = 0, \rho = \mathsf{e}^{\mathcal{I}} \ \mathsf{mod} \ \theta \end{cases}$
```

Fig. 6 A selection function $\mathcal{S}_{\text{LIA}}$ for linear integer arithmetic LIA

eliminated from φ by replacing them with a conjunction of inequalities. Additionally, we assume the signature of integer arithmetic is extended with symbols div^+ and div^-, denoting integer division rounding up and down respectively, and that the language of LIA includes terms of the form $t \ \mathsf{div}^p \ c$, where $p \in \{+, -\}$ and c is a non-zero integer constant. All occurrences of these symbols can be eliminated from any quantifier-free formula φ by repeated applications of the transformation:

$$\varphi[t \ \mathsf{div}^p \ c] \rightsquigarrow \varphi[d] \wedge c \cdot d \approx t \pm^p m \wedge 0 \leq m < c \tag{1}$$

where d and m are distinct fresh variables, and $\pm^p$ is $+$ if p is $+$ and analogously for $-$.

Figure 6 gives a selection function $\mathcal{S}_{\text{LIA}}$ for LIA. The procedure invokes the recursive procedure S_I, which takes as arguments $\mathcal{I}$, $\neg\varphi[\mathbf{k}, \mathbf{e}]$, variables $\mathbf{e}$ that we have yet to incorporate into the substitutions, an integer θ, terms $\mathbf{t}$ found as substitutions for variables from $\mathbf{e}$ so far, and a tuple of symbols $\mathbf{p}$ from $\{+, -\}$ which we refer to as *polarities*. Due to the transformation (1), we may assume without loss of generality that $\neg\varphi[\mathbf{k}, \mathbf{e}]$ contains no instance of integer division. The role of θ will be to capture divisibility relationships through the procedure, where θ is initially 1. The procedure invokes a call to $S_{I0}(\mathcal{I}, \psi, \mathsf{e}_i, \theta)$ which based on the propositionally satisfying assignment for ψ returns a tuple of the form (c, t_i, p_i), where c is a constant, t_i is a term, and p_i is a polarity. The procedure for constructing the term t_i

in the procedure S_{I0} is similar to the procedure S_{R0} in the previous section, where we find the lower bound of the form $c_i \cdot \mathsf{e} \geq \ell_i$ such that the (rational) value $(\frac{\ell_i}{c_i})^{\mathcal{I}}$ is maximal, and similarly for M_u. Additionally, S_{I0} adds a constant ρ to the maximal lower bound (resp. subtracts a constant from the minimal lower bound). This constant ensures that the returned term t_i and e are congruent modulo $\theta \cdot c$ in $\mathcal{I}$, a fact which in part suffices to show the overall function to be model-preserving. It then constructs a *substitution with coefficients* σ of the form $\{c \cdot \mathsf{e}_i \mapsto t_i\}$, where $c \neq 0$. A substitution of this form may be applied to integer terms of the form $c \cdot (d \cdot \mathsf{e}_i + s)$ where $\mathsf{e}_i \notin FV(s)$ and $(c \cdot (d \cdot \mathsf{e}_i + s))\sigma$ is defined as $d \cdot t_i + c \cdot s$. Additionally, we define $(s_1 \bowtie s_2)\sigma$ as $(c \cdot s_1)\sigma \bowtie (c \cdot s_2)\sigma$ for $\bowtie \in \{<, >\}$, and thus we can apply σ to arbitrary LIA-formulas. After constructing σ, the procedure S_I invokes a recursive call where σ is applied to ψ and $(c \cdot \mathbf{t})$, θ is multiplied by c, the term $\theta \cdot t_i$ is appended to $\mathbf{t}$, and p_i is appended to $\mathbf{p}$.

Overall, $\mathcal{S}_{\text{LIA}}$ returns a vector of terms $(\mathbf{t} \mathsf{~div}^{\mathbf{p}}\, \theta)$, that is, integer division applied pairwise to the terms in $\mathbf{t}$ and the constant θ, where $\mathbf{p}$ determines whether this division rounds up or down. When using this selection function in the context of Fig. 1, the instance $\varphi[\mathbf{k}, \mathbf{t} \mathsf{~div}^{\mathbf{p}}\, \theta]$ is added to Γ. Note that our selection function chooses $\mathbf{p}$ such that integer division rounds up for terms coming from lower bounds, and rounds down for terms coming from upper bounds. This choice is not required for correctness, but can reduce the number of instances needed for showing unsatisfiability.

Lemma 6 $\mathcal{S}_{\text{LIA}}$ *is finite for* $\varphi[\mathbf{k}, \mathbf{e}]$.

Proof First, we show only a finite number of tuples are returned by $S_{I0}(\mathcal{I}, (\neg\varphi[\mathbf{k}, \mathbf{e}])\sigma, \mathsf{e}, \theta)$ for any $\mathcal{I}, \sigma, \mathsf{e}$ and finite θ. Let A be the set of atoms occurring in $\varphi[\mathbf{k}, \mathsf{e}_i]\sigma$. The literals in satisfying assignments of $(\neg\varphi[\mathbf{k}, \mathbf{e}])\sigma$ are over these atoms. Let $\{a_1 \cdot \mathsf{e} > t_1, \ldots, a_n \cdot \mathsf{e} > t_n, b_1 \cdot \mathsf{e} < s_1, \ldots, b_m \cdot \mathsf{e} < s_m\}$ be the set of atoms that are in solved form with respect to e and are equivalent to the atoms of A that contain e, where we have $\mathsf{e}_i \notin FV(t_1, \ldots, t_n, s_1, \ldots, s_m)$ and $a_1 > 0, \ldots, a_n > 0, b_1 > 0, \ldots, b_m > 0$. The tuples returned by the call to $S_{I0}(\mathcal{I}, (\neg\varphi[\mathbf{k}, \mathbf{e}])\sigma, \mathsf{e}, \theta)$ are in the set:

$$\begin{aligned}
&\{(a_i, t_i - \rho, -) \mid 1 \leq i \leq n, 0 \leq \rho < \theta \cdot a_i\} \cup \\
&\{(a_i, t_i + 1 + \rho, +) \mid 1 \leq i \leq n, 0 \leq \rho < \theta \cdot a_i\} \cup \\
&\{(b_i, s_i - 1 - \rho, -) \mid 1 \leq i \leq m, 0 \leq \rho < \theta \cdot b_i\} \cup \\
&\{(b_i, s_i + \rho, +) \mid 1 \leq i \leq m, 0 \leq \rho < \theta \cdot b_i\} \cup \{(1, \rho, +) \mid 0 \leq \rho < \theta\}
\end{aligned}$$

Notice that since atoms can appear positively or negatively in M, we consider two possible tuples for each literal in the above set. Since θ is finite, there are a finite number of tuples of this form. Since there are only a finite number of recursive calls to S_I within $\mathcal{S}_{\text{LIA}}$, and each call modifies $\mathbf{t}$ based a finite number of possible tuples coming from the set above, the set of possible return values of $\mathcal{S}_{\text{LIA}}$ is finite, and thus it is finite for $\varphi[\mathbf{k}, \mathbf{e}]$. □

Lemma 7 *If* $\mathcal{I}$ *is a model for* LIA *and for quantifier-free* ψ, $\theta \geq 1$, *and* $S_{I0}(\mathcal{I}, \psi, \mathsf{e}, \theta) = (c, t, p)$, *then:*

1. *c1.]* $(c \cdot \mathsf{e})^{\mathcal{I}} \equiv_{\theta \cdot c} t^{\mathcal{I}}$, *and*
2. $\mathcal{I} \models \psi\{c \cdot \mathsf{e} \mapsto t\}$.

Proof We first show part 7. When $n > 0$ and $S_{I0}(\mathcal{I}, \psi, \mathsf{e}, \theta) = (c_i, \ell_i + \rho, +)$, we have

$$(\ell_i + \rho)^{\mathcal{I}} \equiv_{\theta \cdot c_i} \left(\ell_i + (c_i \cdot \mathsf{e} - \ell_i)^{\mathcal{I}} \mathsf{~mod~} (\theta \cdot c_i)\right)^{\mathcal{I}} \equiv_{\theta \cdot c_i} (c_i \cdot \mathsf{e})^{\mathcal{I}}.$$

When $m > 0$ and $S_{I0}(\mathcal{I}, \psi, \mathsf{e}, \theta) = (d_j, u_j - \rho, -)$, we have

$$\left(u_j - \rho\right)^{\mathcal{I}} \equiv_{\theta \cdot d_j} \left(u_j - \left(u_j - d_j \cdot \mathsf{e}\right)^{\mathcal{I}} \text{ mod } \left(\theta \cdot d_j\right)\right)^{\mathcal{I}} \equiv_{\theta \cdot d_j} \left(d_j \cdot \mathsf{e}\right)^{\mathcal{I}}.$$

When $n = 0$, $m = 0$, and $S_{I0}(\mathcal{I}, \psi, \mathsf{e}, \theta) = (1, \rho, +)$, we have

$$\rho^{\mathcal{I}} \equiv_{\theta \cdot 1} \left(\mathsf{e}^{\mathcal{I}} \text{ mod } \theta\right)^{\mathcal{I}} \equiv_{\theta \cdot 1} (1 \cdot \mathsf{e})^{\mathcal{I}}$$

To show part 7, we first focus on the case where $n > 0$ and $S_{I0}(\mathcal{I}, \psi, \mathsf{e}, \theta) = (c_i, \ell_i + \rho, +)$. We have that $\rho = (c_i \cdot \mathsf{e} - \ell_i)^{\mathcal{I}}$ mod $(\theta \cdot c_i)$. Let M be a set of literals of the form described in the body of $S_{I0}(\mathcal{I}, \psi, \mathsf{e})$. We show that $\mathcal{I}$ satisfies each literal in $M\sigma$, where $\sigma = \{c_i \cdot \mathsf{e} \mapsto \ell_i + \rho\}$. First, consider an atom in $M_\ell\sigma$ that is equivalent to $(c_j \cdot \mathsf{e} \geq \ell_j)\sigma$ for some $j \in \{1, \ldots, n\}$. This is equivalent to $(c_j \cdot c_i \cdot \mathsf{e} \geq c_i \cdot \ell_j)\sigma$, which is equivalent to $\frac{c_j \cdot c_i}{c_i} \cdot (\ell_i + \rho) \geq \frac{c_j \cdot c_i}{c_j} \cdot \ell_j$, which is satisfied by $\mathcal{I}$ since $(\frac{\ell_i}{c_i})^{\mathcal{I}} \geq (\frac{\ell_j}{c_j})^{\mathcal{I}}$ by our selection of $(c_i, \ell_i + \rho)$ and since $\rho \geq 0$. Second, consider the atom in $M_u\sigma$ that is equivalent to $(d_j \cdot \mathsf{e} \leq u_j)\sigma$ for some $j \in \{1, \ldots, m\}$. Let $\rho' = (c_i \cdot \mathsf{e} - \ell_i)^{\mathcal{I}}$, which is greater than 0 since $\mathcal{I}$ satisfies $(c_i \cdot \mathsf{e} \geq \ell_i)$. Since $(c_i \cdot \mathsf{e})^{\mathcal{I}} = (\ell_i + \rho')^{\mathcal{I}}$, we have that $\mathcal{I}$ satisfies $(d_j \cdot \mathsf{e} \leq u_j)\{c_i \cdot \mathsf{e} \mapsto \ell_i + \rho'\}$, which is equivalent to $(d_j \cdot (\ell_i + \rho') \leq c_i \cdot u_j)$. Since $\rho = \rho'$ mod $(\theta \cdot c_i) \leq \rho'$, we have that $\mathcal{I}$ also satisfies $(d_j \cdot (\ell_i + \rho) \leq c_i \cdot u_j)$, which is $(d_j \cdot \mathsf{e} \leq u_j)\sigma$. Finally, $\mathcal{I}$ satisfies $M_c\sigma$ as $M_c\sigma = M_c$ and $\mathcal{I} \models M_c$. Thus, $\mathcal{I}$ satisfies $M\sigma$, which entails $\psi\sigma$. The case when $m > 0$ and $S_{I0}(\mathcal{I}, \psi, \mathsf{e}, \theta) = (d_j, u_j - \rho, -)$ is symmetric. When $n = 0$, $m = 0$, and $S_{I0}(\mathcal{I}, \psi, \mathsf{e}) = (1, \rho, +)$, the assignment M does not contain e, and thus $\mathcal{I}$ satisfies $M\{c \cdot \mathsf{e} \mapsto \rho\} = M$ and $\psi\{c \cdot \mathsf{e} \mapsto \rho\}$. □

Lemma 8 *Each recursive call to $S_I(\mathcal{I}, \psi, (\mathsf{e}_i, \ldots, \mathsf{e}_n), \theta, (t_1, \ldots, t_{i-1}), \mathbf{p})$ within a call to $\mathcal{S}_{\text{LIA}}(\mathcal{I}, \Gamma, \neg\varphi[\mathbf{k}, \mathbf{e}], (\mathsf{e}_1, \ldots, \mathsf{e}_n))$ is such that:*

1. *$\theta \mid t_j^{\mathcal{I}}$ for each $1 \leq j < i$, and*
2. *$\mathcal{I} \models \psi$ and ψ is equivalent to $\neg\varphi[\mathbf{k}, \mathbf{e}]\{\theta \cdot \mathsf{e}_1 \mapsto t_1\} \cdot \ldots \cdot \{\theta \cdot \mathsf{e}_{i-1} \mapsto t_{i-1}\}$.*

Proof Both statements clearly hold for the initial call to S_I in the body of $\mathcal{S}_{\text{LIA}}$. Now, assume both statements hold for some call to $S_I(\mathcal{I}, \psi, (\mathsf{e}_i, \mathbf{e}'), \theta, (t_1, \ldots, t_{i-1}), \mathbf{p})$, and assume $(c, t_i, p_i) = S_{I0}(\mathcal{I}, \psi, \mathsf{e}_i, \theta)$. We show that both statements hold for the call to $S_I(\mathcal{I}, \psi\sigma, \mathbf{e}', \theta \cdot c, ((c \cdot t_1)\sigma, \ldots, (c \cdot t_{i-1})\sigma, \theta \cdot t_i), (\mathbf{p}, p_i))$, where $\sigma = \{c \cdot \mathsf{e}_i \mapsto t_i\}$.

To show part 1, we have from Lemma 7 part 1 that:

$$(c \cdot \mathsf{e}_i)^{\mathcal{I}} \equiv_{\theta \cdot c} t_i^{\mathcal{I}} \tag{2}$$

Consider a t_j where $1 \leq j < i$, which by our assumption is such that $\theta \mid t_j^{\mathcal{I}}$, and thus $\theta \cdot c \mid (c \cdot t_j)^{\mathcal{I}}$. By (2), we have that $\theta \cdot c \mid ((c \cdot t_j)\sigma)^{\mathcal{I}}$. Also by (2), we have that $c \mid t_i^{\mathcal{I}}$, and thus $\theta \cdot c \mid (\theta \cdot t_i)^{\mathcal{I}}$.

To show part 2, by our assumption, $\mathcal{I} \models \psi$ and thus by Lemma 7 part 7 we have that $\mathcal{I} \models \psi\sigma$. By our assumption, ψ is equivalent to $\neg\varphi[\mathbf{k}, \mathbf{e}]\{\theta \cdot \mathsf{e}_1 \mapsto t_1\} \cdot \ldots \cdot \{\theta \cdot \mathsf{e}_{i-1} \mapsto t_{i-1}\}$. Thus, $\psi\sigma$ is equivalent to $\neg\varphi[\mathbf{k}, \mathbf{e}]\{(\theta \cdot c) \cdot \mathsf{e}_1 \mapsto (c \cdot t_1)\sigma\} \cdot \ldots \cdot \{(\theta \cdot c) \cdot \mathsf{e}_{i-1} \mapsto (c \cdot t_{i-1})\sigma\} \cdot \{(\theta \cdot c) \cdot \mathsf{e}_i \mapsto \theta \cdot t_i\}$. Thus, the lemma holds. □

Lemma 9 *$\mathcal{S}_{\text{LIA}}$ is model-preserving for $\varphi[\mathbf{k}, \mathbf{e}]$.*

Proof Assume $\mathcal{S}_{\text{LIA}}(\mathcal{I}, \Gamma, \neg\varphi[\mathbf{k}, \mathbf{e}], \mathbf{e}) = \mathbf{t}$, where $\mathbf{e} = (\mathsf{e}_1, \ldots, \mathsf{e}_n)$ and $\mathbf{t} = (t_1, \ldots, t_n)$. By Lemma 8 and the definition of $\mathcal{S}_{\text{LIA}}$, there is a θ such that for each $i = 1, \ldots, n$, term t_i is of the form s_i div^p θ where $\theta \mid s_i^{\mathcal{I}}$, and $\mathcal{I} \models (\neg\varphi[\mathbf{k}, \mathbf{e}])\{\theta \cdot \mathsf{e}_1 \mapsto s_1\} \cdot \ldots \cdot \{\theta \cdot \mathsf{e}_n \mapsto s_n\}$. Thus, $\mathcal{I}$ satisfies $(\neg\varphi[\mathbf{k}, \mathbf{e}])\{\mathbf{e} \mapsto \mathbf{t}\} = \neg\varphi[\mathbf{k}, \mathbf{t}]$, and thus $\mathcal{S}_{\text{LIA}}$ is model-preserving for $\varphi[\mathbf{k}, \mathbf{e}]$. □

Theorem 3 *$\mathcal{P}_{\mathcal{S}_{\text{LIA}}}$ is a sound and complete procedure for determining the* LIA*-satisfiability of* $\exists \mathbf{k}\, \forall \mathbf{x}\, \varphi[\mathbf{k}, \mathbf{x}]$.

Proof By Theorem 1, Lemma 2, Lemma 6 and Lemma 9. □

Example 6 To demonstrate a case involving a substitution with coefficients, consider the formula $\forall xy\,(2 \cdot x < a \vee x + 3 \cdot y < b)$ whose negation is $2 \cdot e_1 \geq a \wedge e_1 + 3 \cdot e_2 \geq b$. A possible run of $\mathcal{P}_{\mathcal{S}_{\text{LIA}}}$ on this input is as follows.

			$S_{I0}(\mathcal{I}, \psi, \mathsf{e}, \theta)$					
#	Γ	Γ'	e	θ	M_ℓ	Return	$\mathbf{t}[\mathbf{k}]$	Add to Γ
1	sat	sat	e_1	1	$\{\underline{2 \cdot e_1 \geq a}, \ldots\}$	$(2, a, +)$		
			e_2	2	$\{\underline{6 \cdot e_2 \geq 2 \cdot b - a}\}$	$(6, 2 \cdot b - a, +)$	$(6 \cdot a, 4 \cdot b - 2 \cdot a)\ \mathsf{div}^+\ 12$	ψ_1
2	unsat							

We assume $\rho = 0$ for all calls to S_{I0} in this run. Applying the substitution $\{2 \cdot e_1 \mapsto a\}$ to $e_1 + 3 \cdot e_2 \geq b$ results in the bound $6 \cdot e_2 \geq 2 \cdot b - a$ for e_2. We add to Γ an instance, call it ψ_1, which is equivalent to $2 \cdot ((6 \cdot a)\ \mathsf{div}^+\ 12) < a \vee (6 \cdot a)\ \mathsf{div}^+\ 12 + 3 \cdot ((4 \cdot b - 2 \cdot a)\ \mathsf{div}^+\ 12) < b$, which after eliminating integer division is:

$$(2 \cdot k_1 < a \vee k_1 + 3 \cdot k_2 < b) \wedge 12 \cdot k_1 \approx 6 \cdot a + m_1 \wedge 0 \leq m_1 < 12 \wedge \\ 12 \cdot k_2 \approx (4 \cdot b - 2 \cdot a) + m_2 \wedge 0 \leq m_2 < 12$$

which is equisatisfiable to:

$$(6 \cdot a + m_1 < 6 \cdot a \vee 12 \cdot b + m_1 + 3 \cdot m_2 < 12 \cdot b) \wedge 0 \leq m_1 < 12 \wedge \\ 0 \leq m_2 < 12$$

which is LIA-unsatisfiable. Thus, $\exists ab\, \forall xy\, (2 \cdot x < a \vee x + 3 \cdot y < b)$ is LIA-unsatisfiable. □

Example 7 To demonstrate a case involving a non-zero value of ρ, consider the formula $\forall xy\,(3 \cdot x + y \not\approx a \vee 0 > y \vee y > 2)$ whose negation is $3 \cdot e_1 + e_2 \approx a \wedge 0 \leq e_2 \wedge e_2 \leq 2$, where $\approx$ denotes the conjunction of non-strict upper and lower bounds. A possible run of $\mathcal{P}_{\mathcal{S}_{\text{LIA}}}$ on this input is as follows.

			$S_{I0}(\mathcal{I}, \psi, \mathsf{e}, \theta)$					
#	Γ	Γ'	e	θ	M_ℓ	Return	$\mathbf{t}[\mathbf{k}]$	Add to Γ
1	sat	sat	e_1	1	$\{\underline{3 \cdot e_1 \geq a - e_2}\}$	$(3, a - e_2, +)$		
			e_2	3	$\{\underline{e_2 \geq 0}\}$	$(1, 0, +)$	$(a, 0)\ \mathsf{div}^+\ 3$	ψ_1
2	sat	sat	e_1	1	$\{\underline{3 \cdot e_1 \geq a - e_2}\}$	$(3, a - e_2, +)$		
			e_2	3	$\{\underline{e_2 \geq 0}\}$	$(1, 1, +)$	$(a - 1, 1)\ \mathsf{div}^+\ 3$	ψ_2
3	sat	sat	e_1	1	$\{\underline{3 \cdot e_1 \geq a - e_2}\}$	$(3, a - e_2, +)$		
			e_2	3	$\{\underline{e_2 \geq 0}\}$	$(1, 2, +)$	$(a - 2, 2)\ \mathsf{div}^+\ 3$	ψ_3
4	unsat							

On the first iteration, we assume that Γ' is satisfied by a model, call it $\mathcal{I}_1$, that interprets all variables as 0, and hence the values chosen for e_1 and e_2 correspond to their maximal lower bounds in $\mathcal{I}_1$, $a - e_2$ and 0 respectively, where in each call to S_{I0} we have $\rho = 0$. The instance ψ_1 added to Γ on this iteration is equivalent to $3 \cdot (a \ \mathsf{div}^+\ 3) \not\approx a$ and implies that $a^{\mathcal{I}} \not\equiv_3 0$ in subsequent models $\mathcal{I}$. Thus, models $\mathcal{I}$ satisfying $3 \cdot e_1 + e_2 \approx a$ are such that $e_2^{\mathcal{I}} \not\equiv_3 0$. On the next iteration, Γ' is satisfied by a model, call it $\mathcal{I}_2$, where the maximal lower bound for e_2 is 0. By the above reasoning and since $\mathcal{I}_2$ satisfies $3 \cdot e_1 + e_2 \approx a$, it must be that $\rho = ((e_2 - 0)^{\mathcal{I}_2} \ \mathsf{mod}\ 3) \neq 0$. Assume $(e_2 - 0)^{\mathcal{I}_2} \equiv_3 1$. The instance ψ_2 is equivalent to $3 \cdot ((a - 1) \ \mathsf{div}^+\ 3) + 1 \not\approx a$, which implies that $a^{\mathcal{I}} \not\equiv_3 1$ in subsequent models $\mathcal{I}$, and hence $e_2^{\mathcal{I}} \not\equiv_3 1$. The instance ψ_3 is equivalent to $3 \cdot ((a - 2) \ \mathsf{div}^+\ 3) + 2 \not\approx a$ and implies that $a^{\mathcal{I}} \not\equiv_3 2$, which together with the two previous instances are T-unsatisfiable. This run shows $\exists a\, \forall x y\, (3 \cdot x + y \not\approx a \vee 0 > y \vee y > 2)$ is LIA-unsatisfiable. □

The procedure $\mathcal{P}_{\mathcal{S}_{\mathrm{LIA}}}$ can be understood to lazily enumerating disjuncts of Cooper's algorithm for quantifier elimination over linear integer arithmetic [16], with minor differences. The algorithm is essentially enumerating a single path of [16] by using the model to select a satisfied case split for each variable over an entire block of quantifiers.[1] Like that approach, the worst-case performance is dependent upon the size of coefficients of monomials, which is manifested in our case by the fact that the number of possible return values of S_{I0} is proportional to the size of θ. While not shown here, our implementation takes steps to reduce the size of θ by factoring out common divisors in θ and the coefficients returned by S_{I0}.

4.1 Comparison to existing approaches

The approach taken in $\mathcal{P}_{\mathcal{S}_{\mathrm{LIA}}}$ is similar to the one taken in Section 2.5 in [13]. The most substantial difference between the two algorithms is that $\mathcal{P}_{\mathcal{S}_{\mathrm{LIA}}}$ implements a variant of Cooper's algorithm while *resolve* in [13] uses the model to guide an execution of the Omega test [47]. The most similar aspects of the approaches are the computation of a feasible ρ and the computation of the d values in the *grey* shadow cases of *resolve*. These differ in that a different d value is selected to ensure separation between each upper bound and the greatest lower bound in a *projection* whereas ρ is selected using the current value of $c_i \cdot e$ (the selection of d is agnostic to e in our parlance) to ensure all bounds are satisfied by a single instantiation.

5 Instantiation for LIRA-formulas

The methods in the previous two sections can be used in part for the class of LIRA-formulas with one alternation where constraints are over both real and integer variables. For convenience, we assume the signature of LIRA is extended with conversion functions $\mathsf{to\_int}^+$ and $\mathsf{to\_int}^-$ of sort $\mathsf{Real} \rightarrow \mathsf{Int}$, denoting the result of rounding its (real) argument to an integer. All occurrences of these symbols can be eliminated from quantifier-free constraints by the transformation:

$$\varphi[\mathsf{to\_int}^p(t)] \rightsquigarrow \varphi[i] \wedge 0 \leq \pm^p(i - t) < 1 \qquad (3)$$

where i is a fresh constant of type Int, and $\pm^p$ is $+$ if p is $+$ and analogously for $-$.

[1] In the parlance of [16], $\mathcal{P}_{\mathcal{S}_{\mathrm{LIA}}}$ selects a feasible j value using the calculation of ρ and avoids introducing the $F_{\pm\infty}$ cases by introducing the no bounds case ($n = 0, m = 0$) and always favoring bounds when one exists.

```
$\mathcal{S}_{\text{LIRA}}(\mathcal{I}, \Gamma, \neg\varphi[\mathbf{k}, \mathbf{e_r}, \mathbf{e_i}], \mathbf{e})$:
    Let $\mathbf{t_r} = S_R(\mathcal{I}, \neg\varphi[\mathbf{k}, \mathbf{e_r}, \mathbf{e_i}], \mathbf{e_r}, ())$.
    Let $(\mathcal{I}', \varphi') = \text{to\_lia}(\mathcal{I}, \neg\varphi[\mathbf{k}, \mathbf{t_r}, \mathbf{e_i}])$. Let $\mathbf{t_i} = \text{to\_lira}(S_I(\mathcal{I}', \varphi', \mathbf{e_i}, 1, (), ()))$.
    Return $(\mathbf{t_r}\{\mathbf{e_i} \rightarrow \mathbf{t_i}\}, \mathbf{t_i})$.
```

Fig. 7 A selection function $\mathcal{S}_{\text{LIRA}}$ for quantifier-free LIRA-formula $\varphi[\mathbf{k}, \mathbf{e_r}, \mathbf{e_i}]$, where $\mathbf{e_r}$ are real variables and $\mathbf{e_i}$ are integer variables. In this procedure, to_lia($\mathcal{I}$, ψ) denotes the result of casting LIRA-formula ψ to a LIA one, and $\mathcal{I}'$ is an extension of $\mathcal{I}$ (see Example 8)

Figure 7 gives a selection function $\mathcal{S}_{\text{LIRA}}$ for LIRA, where by the above transformation we may assume without loss of generality that $\neg\varphi[\mathbf{k}, \mathbf{e_r}, \mathbf{e_i}]$ contains no conversion functions. Real variables $\mathbf{e_r}$ are processed before integer ones $\mathbf{e_i}$. The procedure invokes the selection function S_R for LRA which returns a set of terms $\mathbf{t_r}$. Afterwards, we apply a transformation, denoted to_lia, to the formula $\neg\varphi[\mathbf{k}, \mathbf{t_r}, \mathbf{e_i}]$, which returns a pair $(\mathcal{I}', \varphi')$ where $\mathcal{I}'$ is an extension of $\mathcal{I}$, and φ' is a LIA formula. We construct φ' by replacing each literal L of the form $(\neg)(\mathbf{c_i} \cdot \mathbf{x_i} + \mathbf{c_r} \cdot \mathbf{x_r} \bowtie c)$ in $\neg\varphi[\mathbf{k}, \mathbf{t_r}, \mathbf{e_i}]$ by the literal $(\neg)(\mathbf{c_i} \cdot \mathbf{x_i} + i^p_{\mathbf{c_r} \cdot \mathbf{x_r}} \bowtie c)$ where $\mathbf{x_i}$ are of type Int, $\mathbf{x_r}$ are of type Real, $i^p_{\mathbf{c_r} \cdot \mathbf{x_r}}$ is a fresh variable of type Int and p is one of $\{+, -\}$. We define the interpretation of $i^p_{\mathbf{c_r} \cdot \mathbf{x_r}}$ in $\mathcal{I}'$ to be the result of rounding the value $(\mathbf{c_r} \cdot \mathbf{x_r})^{\mathcal{I}}$ to an integer up if p is $+$ or down if p is $-$. If this variable occurs in a literal L that entails a lower bound on $i^p_{\mathbf{c_r} \cdot \mathbf{x_r}}$, then p is $-$, otherwise p is $+$. Notice that in either case, the interpretation of L remains unchanged by the transformation to_lia. After constructing to_lia($\mathcal{I}$, $(\neg\varphi[\mathbf{k}, \mathbf{t_r}, \mathbf{e_i}])$), the procedure calls the selection function S_I on the resulting interpretation and formula, which returns a tuple of terms, call it $\mathbf{s_i}$. We then obtain a tuple of terms $\mathbf{t}_i$ by applying a second transformation, denoted to_lira, to $\mathbf{s}_i$, which replaces all LIA-variables of the form i^p_t in $\mathbf{s}_i$ with the LIRA-term $\text{to\_int}^p(t)$. Overall, we return the tuple $(\mathbf{t_r}\{\mathbf{e_i} \rightarrow \mathbf{t_i}\}, \mathbf{t_i})$, where integer variables $\mathbf{e_i}$ are replaced by $\mathbf{t_i}$ within the terms $\mathbf{t_r}$ selected for the real variables. When using this selection in the context of the procedure in Fig. 1, the instance $\varphi[\mathbf{k}, \mathbf{t_r}\{\mathbf{e_i} \rightarrow \mathbf{t_i}\}, \mathbf{t_i}]$ is added to Γ.

Example 8 Consider the formula $\forall x$: Real y:Int $(x - 2 \cdot y < b \vee y < a)$ where a and b are free constants of type Real and Int respectively. The negated Skolemized form of this formula is equivalent to $e_1 - 2 \cdot e_2 \geq b \wedge e_2 \geq a$, where e_1 and e_2 are fresh constants. A possible run of $\mathcal{P}_{\mathcal{S}_{\text{LIRA}}}$ on this input is as follows.

#	Γ	Γ'	e	M_ℓ	(S_R, S_I) return	$\mathbf{t}[\mathbf{k}]$	Add to Γ
1	sat	sat	e_1	$\{\underline{e_1 \geq b + 2 \cdot e_2}\}$	$(b + 2 \cdot e_2)$		
			e_2	$\{\underline{e_2 \geq i^+_a}\}$	(i^+_a)	$(b + 2 \cdot \text{to\_int}^+(a), \text{to\_int}^+(a))$	$\bot$
2	unsat						

On the first iteration, we run S_R which returns the term $b+2 \cdot e_2$ for x based on the bound $e_1 \geq b+2 \cdot e_2$. We then call to_lia($\mathcal{I}$, $((b+2 \cdot e_2) - 2 \cdot e_2 \geq b \wedge e_2 \geq a)$), which returns $(\mathcal{I}', \psi')$, where ψ' is equivalent to $e_2 \geq i^+_a$, and $\mathcal{I}'$ interprets i^+_a as $a^{\mathcal{I}}$ was rounded up to an integer value. We subsequently call S_I on this interpretation and formula, which results in the selection of term i^+_a for e_2. Thus, S_I returns the tuple (i^+_a), where applying the transformation to_lira gives us $(\text{to\_int}^+(a))$. Overall, $\mathcal{S}_{\text{LIRA}}$ returns the tuple $(b + 2 \cdot \text{to\_int}^+(a), \text{to\_int}^+(a))$, which we obtain by applying the substitution $\{y \mapsto \text{to\_int}^+(a)\}$ to our value for e_1 and appending our value $\text{to\_int}^+(a)$ for e_2. Applying the substitution $\{x \mapsto b+2\cdot\text{to\_int}^+(a), y \mapsto \text{to\_int}^+(a)\}$ to our input results in the formula $(b+2\cdot\text{to\_int}^+(a)-2\cdot\text{to\_int}^+(a) < b \vee \text{to\_int}^+(a) < a)$, which after simplification is $(\text{to\_int}^+(a) < a)$. After eliminating conversion functions, this formula is $i < a \wedge 0 \leq i - a < 1$ where i is a fresh integer variable, which is LIRA-unsatisfiable. This run shows that $\exists a\colon \text{Real}\, b\colon \text{Int}\, \forall x\colon \text{Real}\, y\colon \text{Int}\, (x - 2\cdot y < b \vee y < a)$ is LIRA-unsatisfiable. □

It is straightforward that the use of this selection function in Fig. 1 gives a sound procedure for quantified LIRA due to Corollaries 1 and 2, and the transformation (3) preserves equivalence (up to variables $\mathbf{k}$). We do not provide a formal proof of completeness for this approach, although we note that quantifier elimination is possible for this fragment [65].

6 Boolean structure and nested quantification

This section presents a novel technique for establishing the T-satisfiability of formulas with Boolean structure and nested quantification. The technique generalizes the instantiation-based procedure as described in Sect. 2, and can be integrated within the solving architecture used by SMT solvers.

In the following, we show an approach for determining the T-satisfiability of closed T-formula $(\neg)\varphi$. Without loss of generality, we assume φ is a formula from the following grammar:

$$\varphi := \neg\forall\mathbf{x}\,\varphi \mid G \mid \varphi_1 \vee \cdots \vee \varphi_m \tag{4}$$

where G is quantifier-free. In other words, all quantification in our input occurs as a (negated) child of a disjunction. Notice this grammar is effectively a generalization of prenex normal form, since all formulas in prenex normal form are equivalent to a formula in a subset of the above grammar which restricts $m = 1$.[2]

At its core, our approach for establishing the satisfiability of φ is the following. Let $\forall\mathbf{y}\,\psi[\mathbf{k},\mathbf{y}]$ be a sub-formula of our input, where this formula may occur beneath any number of negations and ψ is quantifier-free. Using the procedure in Fig. 1, construct a set of instances $\{\psi[\mathbf{k},\mathbf{t}_1[\mathbf{k}]],\ldots,\psi[\mathbf{k},\mathbf{t}_n[\mathbf{k}]]\}$ that is either unsatisfiable, or that collectively entail $\forall\mathbf{y}\,\psi[\mathbf{k},\mathbf{y}]$. Note that the free variables $\mathbf{k}$ of this sub-formula are considered to be existentially quantified implicitly here. If this set is unsatisfiable, replace $\forall\mathbf{y}\,\psi[\mathbf{k},\mathbf{y}]$ in the input by $\bot$. Otherwise, replace $\forall\mathbf{y}\,\psi[\mathbf{k},\mathbf{y}]$ in the input by the (quantifier-free) formula $\psi[\mathbf{k},\mathbf{t}_1[\mathbf{k}]]\wedge\cdots\wedge\psi[\mathbf{k},\mathbf{t}_n[\mathbf{k}]]$. Repeat this process until our input is replaced by a quantifier-free formula. In this section, we describe an approach that is based on the above reasoning, but is amenable to the standard solving architecture used by SMT solvers. In particular, the approach will be based on incrementally constructing a set of quantifier-free formulas Γ that approximate the input φ, where this set is periodically checked for T-satisfiability.

We begin with the following preliminaries. For each closed quantified T-formula, we associate a Boolean variable A called the *positive guard* of $\forall\mathbf{x}\,\varphi$, and unique set of Skolem variables $\mathbf{e}$ of the same sort as $\mathbf{x}$. We write $(A,\mathbf{e}) \leftrightharpoons \forall\mathbf{x}\,\varphi$ to denote that A and $\mathbf{e}$ are associated with $\forall\mathbf{x}\,\varphi$. We write $\lfloor\varphi\rfloor$ for the result of replacing in φ all closed quantified formulas (not occurring beneath other quantifiers in φ) with their corresponding positive guards. We write $\mathcal{A}(\varphi)$ to denote the set of positive guards in φ. Our approach maintains an evolving set of formulas Γ. We add formulas of the form $A \Rightarrow \phi$ to Γ, where ϕ is a quantifier-free formula that is entailed by $\forall\mathbf{x}\,\varphi$ and $(A,\mathbf{e}) \leftrightharpoons \forall\mathbf{x}\,\varphi$. We call such formulas *guarded instances*. We

[2] For example, notice that $\neg\forall x_1\,\exists x_2\,\forall x_3\,\exists x4\,\varphi$ is equivalent to $\neg\forall x_1\,\neg\forall x_2\,\neg\forall x_3\,\neg\forall x_4\,\neg\varphi$.

write $\lfloor \psi \rfloor_\Gamma$ to denote the result of replacing in $\lfloor \psi \rfloor$ each positive guard A by the conjunction of formulas on the right hand sides of instances from Γ that are guarded by A. Conceptually, the formula $\lfloor \psi \rfloor_\Gamma$ will correspond to the current quantifier-free approximation of ψ in Γ in our approach.

Example 9 Let φ be $\neg\forall x\, P(x) \vee \neg\forall y\, R(y) \vee \neg\forall z\, Q(z) \vee G$ where G is quantifier-free, let $A_1 \leftrightharpoons (\forall x\, P(x), e_1)$, $A_2 \leftrightharpoons (\forall y\, R(y), e_2)$ and $A_3 \leftrightharpoons (\forall z\, Q(z), e_3)$, and let Γ be $\{A_1 \Rightarrow P(a), A_2 \Rightarrow R(b), A_2 \Rightarrow R(c)\}$. Then, $\lfloor\varphi\rfloor$ is $\neg A_1 \vee \neg A_2 \vee \neg A_3 \vee G$, $\mathcal{A}(\lfloor\varphi\rfloor) = \{A_1, A_2, A_3\}$, and $\lfloor\varphi\rfloor_\Gamma$ is $\neg P(a) \vee \neg(R(b) \wedge R(c)) \vee \neg\top \vee G$. □

To determine the T-satisfiability of a closed T-formula $\forall \mathbf{x}\, \varphi[\mathbf{x}]$, we use the procedure solve_T in Fig. 8. This procedure incrementally refines an approximation of $\forall \mathbf{x}\, \varphi[\mathbf{x}]$, given by set Γ, until it is found to be (un)satisfiable in T. The procedure first invokes the recursive subprocedure CEGQI_T, which takes as input a set Γ initially containing the positive guard A_0 of $\forall \mathbf{x}\, \varphi[\mathbf{x}]$, and A_0 itself. This subprocedure adds formulas to Γ in Step 3 until either Γ is T-unsatisfiable (Step 1), in which case the input is unsatisfiable, or otherwise the procedure saturates (Step 2), in which case the input is satisfiable.

Formulas are added to Γ based on the recursive procedure rec_T. Recall that our input formula $\forall \mathbf{x}\, \varphi[\mathbf{x}]$ is a formula having a tree-like structure built from grammar (4). At a high level, the procedure rec_T returns a guarded instance of some quantified sub-formula in this tree whose satisfiability is yet to be determined, if one exists. In detail, this function takes as arguments Γ and the positive guard A of a quantified formula $\forall \mathbf{y}.\psi[\mathbf{k}, \mathbf{y}]$, called initially with $A = A_0$. It first constructs the formula $\phi[\mathbf{k}, \mathbf{e}] = \lfloor \psi[\mathbf{k}, \mathbf{e}] \rfloor_\Gamma$, which represents an approximation of the formula $\psi[\mathbf{k}, \mathbf{e}]$ under the assumption of the current instances in Γ. If the current set of instances $\lfloor A \rfloor_\Gamma$ that are guarded by A and the negation of this formula are unsatisfiable, then it is the case that $\forall \mathbf{y}\, \psi[\mathbf{k}, \mathbf{y}]$ is equivalent to $\lfloor A \rfloor_\Gamma$ and the procedure returns the empty set. Otherwise, we consider the direct children of ψ, i.e. those whose positive guard A' occurs in $\mathcal{A}(\lfloor \psi[\mathbf{k}, \mathbf{e}] \rfloor)$. If the recursive call to rec_T returns a guarded instance for some child A', then the procedure returns that instance. Otherwise, it returns a guarded instance $A \Rightarrow \phi[\mathbf{k}, \mathbf{t}[\mathbf{k}]]$, where $\mathcal{I}$ is a model of theory T, $\lfloor A \rfloor_\Gamma$ and $\neg\phi[\mathbf{k}, \mathbf{e}]$, and terms $\mathbf{t}[\mathbf{k}]$ are chosen by the selection function $\mathcal{S}_T$ for theory T.

```
solve_T(∀x φ[x]):
    Return CEGQI_T({⌊∀xφ[x]⌋}, ⌊∀xφ[x]⌋)
CEGQI_T(Γ, A_0):
  1. If Γ is T-unsatisfiable, then return "unsat".
  2. If rec_T(Γ, A_0) = ∅, then return "sat".
  3. Otherwise, return CEGQI_T(Γ ∪ rec_T(Γ, A_0), A_0).
rec_T(Γ, A), where (A, e) ⥦ ∀y.ψ[k, y]:
    Let φ[k, e] = ⌊ψ[k, e]⌋_Γ.
    If ⌊A⌋_Γ ∧ ¬φ[k, e] is T-unsatisfiable, then return ∅.
    If there exists an A′ ∈ 𝒜(⌊ψ[k, e]⌋) such that rec_T(Γ, A′) ≠ ∅, then return rec_T(Γ, A′).
    Otherwise,
        Let 𝓘 be a model of T and ⌊A⌋_Γ ∧ ¬φ[k, e], and let t[k] = 𝒮_T(𝓘, Γ, ¬φ[k, e], e).
        Return {A ⇒ φ[k, t[k]]}.
```

Fig. 8 Abstract procedure solve for establishing the T-satisfiability of $\forall \mathbf{x}\, \varphi[\mathbf{x}]$, which calls a counterexample-guided approach for quantifier instantiation CEGQI_T. Instances are added to Γ based on a selection function $\mathcal{S}_T$ for theory T. This procedure generalizes the one in Fig. 1

The correctness of this procedure relies on the following facts, where recall from Sect. 1.1 two formulas are equivalent up to $\mathbf{k}$ if the are satisfied by the same set of models when restricted to the interpretation of variables from $\mathbf{k}$.

Lemma 10 *Let* $(A, \mathbf{e}) \leftrightharpoons \forall \mathbf{y}\, \psi[\mathbf{k}, \mathbf{y}]$.

1. *If* $\mathsf{rec}_T(\Gamma, A)$ *returns* $\{A \Rightarrow \phi[\mathbf{k}, \mathbf{t}]\}$*, then* $\phi[\mathbf{k}, \mathbf{t}]$ *is equivalent to* $\psi[\mathbf{k}, \mathbf{t}]$ *up to* $\mathbf{k}$.
2. *If* $\lfloor A \rfloor_\Gamma \wedge \lfloor \neg\psi[\mathbf{k}, \mathbf{e}] \rfloor_\Gamma$ *is* T*-unsat, then* $\forall \mathbf{y}\, \psi[\mathbf{k}, \mathbf{y}]$ *is equivalent to* $\lfloor A \rfloor_\Gamma$ *up to* $\mathbf{k}$.

Proof We prove both facts simultaneously by induction on the structure of ψ.

(Base case) When $\psi[\mathbf{k}, \mathbf{y}]$ is quantifier-free, then $\lfloor \psi[\mathbf{k}, \mathbf{e}] \rfloor_\Gamma = \psi[\mathbf{k}, \mathbf{e}]$ and $\mathsf{rec}_T(\Gamma, A)$ returns only sets of the form $\{A \Rightarrow \psi[\mathbf{k}, \mathbf{t}[\mathbf{k}]]\}$. Thus, part 1 holds, and moreover we have that $\forall \mathbf{y}\, \psi[\mathbf{k}, \mathbf{y}] \models_T \lfloor A \rfloor_\Gamma$. To show part 2, assume $\lfloor A \rfloor_\Gamma \wedge \lfloor \neg\psi[\mathbf{k}, \mathbf{e}] \rfloor_\Gamma$ is T-unsatisfiable, or in other words $\lfloor A \rfloor_\Gamma \models_T \psi[\mathbf{k}, \mathbf{e}]$. Since $\mathbf{e}$ does not occur in $\lfloor A \rfloor_\Gamma$, we have that $\lfloor A \rfloor_\Gamma \models_T \forall \mathbf{y}\, \psi[\mathbf{k}, \mathbf{y}]$, and thus $\forall \mathbf{y}\, \psi[\mathbf{k}, \mathbf{y}]$ is equivalent to $\lfloor A \rfloor_\Gamma$ up to $\mathbf{k}$.

(Inductive case) When $\psi[\mathbf{k}, \mathbf{y}]$ is $\neg\forall \mathbf{x}_1\, \psi_1[\mathbf{k}_1, \mathbf{x}_1] \vee \cdots \vee \neg\forall \mathbf{x}_m\, \psi_m[\mathbf{k}_m, \mathbf{x}_m]$, for $i = 1, \ldots, m$, let $(A_i, \mathbf{e}_i) \leftrightharpoons \forall \mathbf{x}_i\, \psi_i[\mathbf{k}_i, \mathbf{x}_i]$ if $\mathbf{x}_i$ is non-empty. To show part 1, assume $\mathsf{rec}_T(\Gamma, A)$ returns $\{A \Rightarrow \phi[\mathbf{k}, \mathbf{t}]\}$, where $\phi[\mathbf{k}, \mathbf{t}]$ is $\lfloor \psi[\mathbf{k}, \mathbf{t}] \rfloor_\Gamma$. For each $i = 1, \ldots, m$ where $\mathbf{x}_i$ is non-empty, by definition of rec_T it must be that $\mathsf{rec}_T(\Gamma, A_i)$ returns $\emptyset$, and thus $\lfloor A_i \rfloor_\Gamma \wedge \lfloor \neg\psi_i[\mathbf{k}_i, \mathbf{e}_i] \rfloor_\Gamma$ is T-unsatisfiable. Thus, by part 2 of the induction hypothesis, we have that $\lfloor A_i \rfloor_\Gamma$ is equivalent to $\forall \mathbf{x}_i\, \psi_i[\mathbf{k}_i, \mathbf{x}_i]$ up to $\mathbf{k}_i$ where $\mathbf{k}_i$ is a subset of $\mathbf{k}$. Thus, $\lfloor \psi[\mathbf{k}, \mathbf{t}] \rfloor_\Gamma$ is equivalent to $\psi[\mathbf{k}, \mathbf{t}]$ up to $\mathbf{k}$, and thus part 1 holds. To show part 2, we have by part 1 that $\forall \mathbf{y}\, \psi[\mathbf{k}, \mathbf{y}] \models_T \lfloor A \rfloor_\Gamma$. When $\lfloor A \rfloor_\Gamma \wedge \lfloor \neg\psi[\mathbf{k}, \mathbf{e}] \rfloor_\Gamma$ is T-unsatisfiable, $\forall \mathbf{y}\, \psi[\mathbf{k}, \mathbf{y}]$ is equivalent to $\lfloor A \rfloor_\Gamma$ up to $\mathbf{k}$ for the same reasons as in the base case. □

Theorem 4 *Assume the satisfiability of quantifier-free* T*-formulas is decidable, and a selection function* $\mathcal{S}_T$ *exists that is finite and monotonic.*

1. *If* $\mathsf{solve}_T(\forall \mathbf{x}\, \varphi[\mathbf{x}])$ *returns "unsat", then* $\forall \mathbf{x}\, \varphi[\mathbf{x}]$ *is* T*-unsatisfiable.*
2. *If* $\mathsf{solve}_T(\forall \mathbf{x}\, \varphi[\mathbf{x}])$ *returns "sat", then* $\forall \mathbf{x}\, \varphi[\mathbf{x}]$ *is* T*-satisfiable.*
3. $\mathsf{solve}_T(\forall \mathbf{x}\, \varphi[\mathbf{x}])$ *terminates.*

Proof To show 1, let $\lceil \Gamma \rceil$ denote the result of replacing the positive guards in Γ with the quantified formula they respond to. By definition of solve_T, we have that $\lceil \Gamma \rceil$ contains $\forall \mathbf{x}\, \varphi[\mathbf{x}]$, and additionally contains tautologies of the form $\forall \mathbf{x}\, \psi[\mathbf{k}, \mathbf{y}] \Rightarrow \phi[\mathbf{k}, \mathbf{t}]$ where by Lemma 10.1 $\phi[\mathbf{k}, \mathbf{t}]$ is equivalent to $\psi[\mathbf{k}, \mathbf{t}]$ up to $\mathbf{k}$. Thus, $\lceil \Gamma \rceil$ is equivalent to $\forall \mathbf{x}\, \varphi[\mathbf{x}]$. Furthermore, we have that $\lceil \Gamma \rceil \models_T \Gamma$, and Γ is T-unsatisfiable. Thus, $\forall \mathbf{x}\, \varphi[\mathbf{x}]$ is T-unsatisfiable.

To show 2, by definition of solve_T, we have that Γ is T-satisfiable, and $\lfloor A_0 \rfloor_\Gamma \wedge \neg \lfloor \varphi[\mathbf{e}] \rfloor_\Gamma$ is T-unsatisfiable, where $(A_0, \mathbf{e}) \leftrightharpoons \forall \mathbf{x}\, \varphi[\mathbf{x}]$. By Lemma 10.2, we have that $\forall \mathbf{x}\, \varphi[\mathbf{x}]$ is equivalent to $\lfloor A_0 \rfloor_\Gamma$. Since $A_0 \in \Gamma$, we have that $\Gamma \models_T \lfloor A_0 \rfloor_\Gamma$, and thus $\forall \mathbf{x}\, \varphi[\mathbf{x}]$ is satisfied by a model of Γ.

To show 3, all individual steps in the procedure are terminating since the T-satisfiability of quantifier-free T-formulas is decidable. Furthermore, we show that the number of instances added to Γ is finite. Let $\forall \mathbf{y}\, \psi[\mathbf{k}, \mathbf{y}]$ be a formula where $(A, \mathbf{e}) \leftrightharpoons \forall \mathbf{y}\, \psi[\mathbf{k}, \mathbf{y}]$ and for which $\mathsf{rec}_T(\Gamma, A)$ is called. Assume that at least one instance of the form $\{A \Rightarrow \phi[\mathbf{k}, \mathbf{t}_1[\mathbf{k}]]\}$ is added for some quantifier-free T-formula $\phi[\mathbf{k}, \mathbf{e}] = \lfloor \psi[\mathbf{k}, \mathbf{e}] \rfloor_\Gamma$. Since formulas are never removed from Γ, it must be the case that $\lfloor \psi[\mathbf{k}, \mathbf{e}] \rfloor_\Gamma = \lfloor \psi[\mathbf{k}, \mathbf{e}] \rfloor_{\Gamma \cup \Gamma'}$ for all formulas Γ' added in subsequent recursive calls to CEGQI_T. Thus, all instances returned by rec_T guarded by A are of the form $\{A \Rightarrow \phi[\mathbf{k}, \mathbf{t}_i[\mathbf{k}]]\}$ for the same ϕ. Since $\mathcal{S}_T$ is monotonic for $\exists \mathbf{k}\, \forall \mathbf{y}\, \phi[\mathbf{k}, \mathbf{y}]$, we are guaranteed to add a new instance to Γ on each recursive call to CEGQI_T. Thus, since $\mathcal{S}_T$ is finite for $\exists \mathbf{k}\, \forall \mathbf{y}\, \phi[\mathbf{k}, \mathbf{y}]$, only finitely many instances of this form are returned. Since there are only finitely many $\forall \mathbf{y}\, \psi[\mathbf{k}, \mathbf{y}]$ for which rec_T is called on, we have that only finitely many instances are added to Γ, finitely many recursive calls are made to CEGQI_T, and thus $\mathsf{solve}_T(\forall \mathbf{x}\, \varphi[\mathbf{x}])$ terminates. $\square$

By the previous theorem, assuming satisfiability of quantifier-free T-formulas is decidable and a finite and monotonic selection function exists for T, solve_T is a decision procedure for T-formulas containing arbitrary nested quantification.

Example 10 Consider the LIA-formula $\forall x\, \varphi[x]$ where $\varphi[x]$ is $\neg(\forall y\, x > y \vee 0 > y) \vee x < 0$. Let $(A_1, e_1) \leftrightharpoons \forall x\, \varphi[x]$ and let $(A_2, e_2) \leftrightharpoons \forall y\, e_1 > y \vee 0 > y$. We call CEGQI where Γ is initially $\{A_1\}$. A possible run of this procedure is summarized in the table below.

#	Γ?	rec_T A	$\lfloor \neg\psi[\mathbf{e}] \rfloor$	$\lfloor \neg\psi[\mathbf{e}] \rfloor_\Gamma$	$\lfloor A \rfloor_\Gamma \wedge \lfloor \neg\psi[\mathbf{e}] \rfloor_\Gamma$?	$t[\mathbf{k}]$	return	return
1	sat	A_1	$A_2 \wedge e_1 \geq 0$	$\top \wedge e_1 \geq 0$	sat		$\mathsf{rec}_T(\Gamma, A_2)$	
		A_2	$e_1 \leq e_2 \wedge 0 \leq e_2$	$e_1 \leq e_2 \wedge 0 \leq e_2$	sat	(e_1)	$\{A_2 \Rightarrow 0 > e_1\}$	
2	sat	A_1	$A_2 \wedge e_1 \geq 0$	$0 > e_1 \wedge e_1 \geq 0$	unsat		$\emptyset$	“sat”

On the first call to the procedure CEGQI, Γ is T-satisfiable. We call rec_T on Γ and A_1, which first checks the satisfiability of $\lfloor A_1 \rfloor_\Gamma \wedge \lfloor (\forall y\, e_1 > y \vee 0 > y) \wedge e_1 \geq 0 \rfloor_\Gamma$, which is $\top \wedge (\top \wedge e_1 \geq 0)$, which is satisfiable. It then checks if there exists an A' among the positive guards in $\lfloor (\forall y\, e_1 > y \vee 0 > y) \wedge e_1 \geq 0 \rfloor$ for which an instance can be returned. On the call to rec_T where $A' = A_2$, we find that $e_1 \leq e_2 \wedge 0 \leq e_2$ is satisfiable, and there are no positive guards in $\lfloor e_1 > e_2 \vee 0 > e_2 \rfloor$, that is, $\forall y\, e_1 > y \vee 0 > y$ contains no nested quantifiers. We use the selection function for linear integer arithmetic $\mathcal{S}_{\text{LIA}}$ as given in Sect. 4, which given input $e_1 \leq e_2 \wedge 0 \leq e_2$ returns the tuple $t[e_1] = (e_1)$, thus giving the instance $A_2 \Rightarrow 0 > e_1$ which we add to Γ. On the second call to CEGQI_T, we have that $\Gamma = \{A_1, A_2 \Rightarrow 0 > e_1\}$ is satisfiable, and we again call rec_T on Γ and A_1, where now $0 > e_1 \wedge e_1 \geq 0$ is unsatisfiable. This establishes that $\forall x\, \neg(\forall y\, x > y \vee 0 > y) \vee x < 0$ is LIA-satisfiable. $\square$

Example 11 We remark that treating formulas that are not in prenex normal form allows us to avoid unnecessary computation. Consider the LIA-formula $\forall x\, \varphi[x]$, where $\varphi[x]$ is $(\neg(\forall y\, x > y) \vee \neg\forall z\, \psi[x])$, and $\psi[x]$ is some LIA-formula. Let $(A_1, e_1) \leftrightharpoons \forall x\, \varphi[x]$, let $(A_2, e_2) \leftrightharpoons \forall y\, e_1 > y$, and let $(A_3, e_3) \leftrightharpoons \forall z\, \psi[e_1]$. A possible run of this procedure is summarized in the table below.

#	Γ?	rec_T A	$\lfloor \neg\psi[\mathbf{e}] \rfloor$	$\lfloor \neg\psi[\mathbf{e}] \rfloor_\Gamma$	$\lfloor A \rfloor_\Gamma \wedge \lfloor \neg\psi[\mathbf{e}] \rfloor_\Gamma$?	$t[\mathbf{k}]$	return	return
1	sat	A_1	$A_2 \wedge A_3$	$\top \wedge \top$	sat		$\mathsf{rec}_T(\Gamma, A_2)$	
		A_2	$e_1 \leq e_2$	$e_1 \leq e_2$	sat	(e_1)	$\{A_2 \Rightarrow e_1 > e_1\}$	
2	sat	A_1	$A_2 \wedge A_3$	$e_1 > e_1 \wedge \top$	unsat		$\emptyset$	“sat”

The first call to procedure CEGQI_T, adds the formula $A_2 \Rightarrow e_1 > e_1$ to Γ. In the second call to CEGQI_T, within the call to rec_T for $A = A_1$, we find that $\lfloor A_1 \rfloor_\Gamma \wedge \lfloor (\forall y\, e_1 > y) \wedge \forall z\, \psi[e_1] \rfloor_\Gamma$, which is $\top \wedge (e_1 > e_1 \wedge \top)$, is unsatisfiable. Thus, we conclude that $\forall x\, (\neg(\forall y\, x > y) \vee \neg\forall z\, \psi[x])$ is LIA-satisfiable. This was determined regardless of the content of ψ. $\square$

Example 12 Consider the LIA-formula $\forall xy\, \varphi[x, y]$, where $\varphi[x, y]$ is $(\neg(\forall z\, z < x \vee y < z) \vee x < y + 5)$. Let $(A_1, (e_1, e_2)) \leftrightharpoons \forall xy\, \varphi[x, y]$ and let $(A_3, e_3) \leftrightharpoons \forall z\, z < e_1 \vee e_2 < z$. We call CEGQI where Γ is initially $\{A_1\}$. A possible run of this procedure is summarized in the table below.

#	Γ?	rec_T A	$\lfloor\neg\psi[\mathbf{e}]\rfloor$	$\lfloor\neg\psi[\mathbf{e}]\rfloor_\Gamma$	$\lfloor A\rfloor_\Gamma \wedge \lfloor\neg\psi[\mathbf{e}]\rfloor_\Gamma$?	$t[\mathbf{k}]$	Return	Return
1	sat	A_1	$A_3 \wedge e_1 \geq e_2 + 5$	$\top \wedge e_1 \geq e_2 + 5$	sat		$\mathsf{rec}_T(\Gamma, A_3)$	
		A_3	$e_3 \geq e_1 \wedge e_2 \geq e_3$	$e_3 \geq e_1 \wedge e_2 \geq e_3$	sat	(e_2)	$\{A_3 \Rightarrow e_1 > e_2\}$	
2	sat	A_1	$A_3 \wedge e_1 \geq e_2 + 5$	$e_1 > e_2 \wedge e_1 \geq e_2 + 5$	sat		...	
		A_3	$e_3 \geq e_1 \wedge e_2 \geq e_3$	$e_3 \geq e_1 \wedge e_2 \geq e_3$	unsat		$\emptyset$	
		A_1	...	...	...	$(5, 0)$	$\{A_1 \Rightarrow \bot\}$	
2	unsat							"unsat"

On the first call to CEGQI, we find that Γ is satisfiable, and the call to rec_T returns the guarded instance $A_3 \Rightarrow e_1 > e_2$, which we add to Γ. On the second call to CEGQI, we find that Γ is again satisfiable. The call to rec_T for $A = A_1$ first finds that $\Gamma \cup \{A_3 \wedge e_1 \geq e_2 + 5\}$ is also satisfiable, and invokes itself recursively on A_3. The call to rec_T for $A = A_3$ determines that $\lfloor A\rfloor_\Gamma \wedge \lfloor\neg\psi[\mathbf{e}]\rfloor_\Gamma$, which is $e_1 > e_2 \wedge (e_3 \geq e_1 \wedge e_2 \geq e_3)$, is unsatisfiable and thus returns the empty set. By Lemma 10.2, this indicates that $\forall z\, z < e_1 \vee e_2 < z$ is equivalent to $e_1 > e_2$, that is, the conjunction of instances in Γ that are guarded by A_3. Returning to rec_T for $A = A_1$, we add an instance for $\forall xy\, \varphi[x, y]$ where A_3 is replaced by $e_1 > e_2$ in the construction of $\lfloor\neg\varphi[e_1, e_2]\rfloor_\Gamma$, which gives us $e_1 > e_2 \wedge e_1 \geq e_2 + 5$. Applying the selection function $\mathcal{S}_{\text{LIA}}$ as given in Sect. 4 to this formula returns the tuple $(5, 0)$ for (e_1, e_2), thus giving the instance $A_1 \Rightarrow \neg(5 > 0 \wedge 5 \geq 0 + 5)$, which is equivalent to $A_1 \Rightarrow \bot$. We add this instance to Γ, after which we find that it is unsatisfiable, and thus $\forall xy\, \varphi[x, y]$ is LIA-unsatisfiable. □

6.1 Implementation details

In practice, the approach in Fig. 8 can be accomplished by a single instance of an SMT solver. Although not shown here, we additionally associate a second Boolean variable B with each quantified formula, called its *negative guard*. For each quantified formula $\forall \mathbf{x}\, \varphi$ with negative guard B, we add the formula $B \Rightarrow \neg\varphi[\mathbf{e}]$ to Γ. The function rec_T then can be simulated using a decision heuristic in the underlying SAT solver that decides positively on the negative guards of innermost quantified formulas first, and adds instances of quantified formulas only if their negative guards are not propagated to false at decision level 0. A formal description of this technique is the subject of future work.

An alternative strategy to Fig. 8 is to add instances of (top-most) quantified formulas that may have nested quantification. That is, for a quantified formula $\forall x\, \varphi[x]$, we may add instances of the form $A \Rightarrow \varphi[t]$, where φ contains quantified formulas. In such a strategy, virtual terms in t (δ or ∞) that are substituted beneath quantifiers in φ must be treated specially. Adding instances of this form may potentially allow us to discover unsatisfiable instances quicker, but also may introduce many quantified formulas that in turn degrade performance. In the latest version of our implementation, we do not use this strategy. However, an earlier version of our implementation (CVC4 from SMT COMP 2016) makes use of this strategy.

6.2 Comparison to existing approaches

We refer to the treatment of quantified formulas in Fig. 8 as *counterexample-guided quantifier instantiation* [50]. The algorithm is similar to existing instantiation-based approaches used by SMT solvers for quantified formulas [17,25] in that it adds guarded instances of quantified formulas incrementally. Its instance selection is guided by models for the negation of quantified formulas, similar to model-based quantifier instantiation [26]. This approach differs in its scope, in that it primarily targets quantified formulas having uninterpreted functions, whereas the approach described in Fig. 8 targets quantified formulas having no uninterpreted functions. The approach of [26] also differs in that it uses a separate copy of the SMT solver as an oracle for checking the satisfiability of the negation of each quantified formula it instantiates, whereas our approach uses a single instance of the SMT solver for doing these tasks simultaneously in its main solving loop. Other approaches that are specialized for quantified linear arithmetic invoke separate instances of an SMT solver for each quantifier alternation [13,19], and are often restricted to inputs where quantified formulas are in prenex normal form. In contrast, the approach in this section requires only one instance of the SMT solver and may be applied to inputs with Boolean structure in the grammar (4) described at the beginning of this section.

7 Instantiation as a synthesis procedure

The connection between quantifier elimination and synthesis has been shown fruitful in previous work [34]; it is one of our motivations for further improving quantified reasoning modulo theories. The instantiation procedure mentioned in this paper can be used to synthesize functions from certain classes of specifications. Consider (second-order) T-formulas of the form:

$$\exists \mathbf{f}\, \forall \mathbf{x}\, \varphi[\mathbf{f}, \mathbf{x}] \tag{5}$$

where φ is a quantifier-free formula, $\mathbf{x} = (x_1, \ldots, x_n)$ is a tuple of variables of sort τ_i for $i = 1, \ldots, n$, and $\mathbf{f} = (f_1, \ldots, f_m)$ is a tuple of functions of sort $\tau_1 \times \cdots \times \tau_n \rightarrow \tau_j$ for $j = 1, \ldots, m$. We call such formulas *synthesis conjectures*. A synthesis conjecture is *single invocation* (over $\mathfrak{L}$) if it is equivalent to:

$$\exists \mathbf{f}\, \forall \mathbf{x}\, \psi[\mathbf{x}, \mathbf{f}(\mathbf{x})] \tag{6}$$

where $\psi[\mathbf{x}, \mathbf{y}] \in \mathfrak{L}$. That is, functions $\mathbf{f}$ are applied to the tuple $\mathbf{x}$ only. The formula (6) is equivalent to the (first-order) formula $\forall \mathbf{x}\, \exists \mathbf{y}\, \psi[\mathbf{x}, \mathbf{y}]$, whose negation

$$\exists \mathbf{x}\, \forall \mathbf{y}\, \neg\psi[\mathbf{x}, \mathbf{y}] \tag{7}$$

is suitable as an input to Fig. 1. As observed in [50], solutions for single invocation synthesis conjectures can be extracted from an unsatisfiable core of instantiations when proving the unsatisfiability of (7). In particular, let $\mathbf{k}$ be a set of distinct fresh variables of the same sort as $\mathbf{x}$, and say the set $\{\neg\psi[\mathbf{k}, \mathbf{t}_1[\mathbf{k}]], \ldots, \neg\psi[\mathbf{k}, \mathbf{t}_n[\mathbf{k}]]\}$ is T-unsatisfiable where $\mathbf{t}_i = (t_i^1[\mathbf{k}], \ldots, t_i^m[\mathbf{k}])$ for $i = 1, \ldots, n$. Then:

$$1 \leq j \leq m{:} f_j = \lambda\mathbf{x}.\, \mathsf{ite}(\psi[\mathbf{x}, t_n^j[\mathbf{x}]], t_n^j[\mathbf{x}], (\cdots \mathsf{ite}(\psi[\mathbf{x}, t_2^j[\mathbf{x}]], t_2^j[\mathbf{x}], t_1^j[\mathbf{x}]))) \tag{8}$$

is a solution for $\mathbf{f}$ in (6). The instantiation-based procedure in Fig. 1 can be used to discharge (7). It is important to note that the solution (8) does not necessarily belong to the language $\mathfrak{L}$, since there is no restriction on the selection functions for $\mathfrak{L}$ that restricts its return value $\mathbf{t}$ to terms in $\mathfrak{L}$. For example, in some of our approaches to linear real arithmetic, $\mathbf{t}$ may contain a free distinguished constants δ or ∞ which are outside of the typical language of linear real arithmetic. Different selection functions or post-processing may be required based on the restrictions for the solutions to synthesis conjectures.

In this paper, we have devised selection functions $\mathcal{S}$ for linear real and integer arithmetic that are finite and monotonic in Sects. 3 and 4. This implies a sound and complete method for synthesizing tuples of functions whose specification is a single invocation synthesis conjecture over linear real and integer arithmetic.

Example 13 Consider the second-order LIA-formula $\exists f\, \forall xy\, (f(x, y) \geq x \wedge f(x, y) \geq y)$, which states that there exists a function f that is the maximum of its arguments x and y. This formula is equisatisfiable to the first-order LIA-formula $\forall xy\, \exists z\, (z \geq x \wedge z \geq y)$. We apply the instantiation-based procedure in Fig. 1 on the negation of this input, $\exists xy\, \forall z\, \neg(z \geq x \wedge z \geq y)$. The procedure may find the T-unsatisfiable set of instances based on $\{\neg(x \geq x \wedge x \geq y), \neg(y \geq x \wedge y \geq y)\}$. Thus, a solution for f is $\lambda x_1 x_2.\, \mathsf{ite}((x_1 \geq x_1 \wedge x_1 \geq x_2), x_1, x_2)$, which after simplification is $\lambda x_1 x_2.\, \mathsf{ite}(x_1 \geq x_2, x_1, x_2)$. □

8 Experimental evaluation

We have implemented the procedure in the SMT solver CVC4 [6] (version 1.5 pre-release). This section presents an evaluation of this implementation compared against other SMT solvers, first-order theorem provers and synthesis solvers.

We considered all quantified benchmarks over 6 classes in the LRA and LIA logics of the SMT library [7]. The class **keymaera** are verification conditions coming from the Keymaera verification tool [45], **scholl** were used for simplification of non-convex polyhedra in [54], **psyco** were used for weakest precondition synthesis for compiler optimizations in [37], **uauto** correspond to verification conditions in [28], and the **tptp** classes correspond to simple arithmetic conjectures coming from the TPTP library [58]. We also considered a class of benchmarks **sygus** corresponding to first-order formulations of the 71 single-invocation synthesis conjectures taken from the conditional linear integer track of the 2015 edition of the syntax-guided synthesis competition [1]. All benchmarks are in the SMT version 2 format. For comparisons with automated theorem provers, they were converted to the TPTP format by the SMTtoTPTP conversion tool [8]. We remark that all benchmarks consist purely of quantified formulas over linear arithmetic with very little, and in a majority of cases, no quantifier-free content. Of the 7 benchmark classes, only one (the **scholl** class from LRA) had quantified formulas with nested quantification.[3]

The results for the linear real and integer benchmarks are in Tables 1 and 2 respectively. We considered all SMT solvers and theorem provers from the LRA and LIA divisions of SMT COMP 2016, and the TFA division of CASC J8 [59], the latest competitions in the SMT and automated theorem proving communities.[4] We write **cvc4-sc16** and **z3-sc16** to denote solvers from SMT COMP 2016, where CVC4 implements an earlier version of the techniques from this paper and Z3 (version 4.4.1) implements the techniques from [12]. We additionally considered Yices version 2.4.1 for LRA, and Z3 version 4.4.2 which implements

[3] Details can be found at http://cs.uiowa.edu/~ajreynol/FMSD-InstLA.

[4] The solver **cvc4-sc16** won the LRA and LIA divisions of SMT COMP 2016. The TFA division of CASC J8 includes problems that combine arithmetic and uninterpreted functions, where the techniques in this paper are only partially applicable. Vampire won this division, and CVC4 came in 3rd.

Table 1 Results for LRA benchmarks, showing times (in seconds) and benchmarks solved by each solver and configuration over 3 benchmark classes with a 300s timeout

	keymaera (222)		**scholl** (374)		**tptp** (25)		**Total** (621)	
	#	Time	#	Time	#	Time	#	Time
cvc4+lw	222	4.1	369	700.0	25	0.4	616	704.4
cvc4	222	4.8	369	818.3	25	0.4	616	823.5
z3	222	8.1	368	875.4	25	0.9	615	884.5
cvc4+fr	222	4.1	365	672.7	25	0.7	612	677.5
cvc4+nvt	222	4.0	365	684.4	25	0.4	612	688.8
cvc4-sc16	222	5.4	349	1213.4	25	0.6	596	1219.3
z3-sc16	222	8.0	330	1345.9	25	0.8	577	1354.8
Vampire	222	29.9	100	305.9	25	1.4	347	337.2
VeriT	222	4.0	46	504.3	12	0.3	280	508.6
Yices	222	3.9	–	0.0	25	0.4	247	4.2
Princess	202	1068.7	0	0.0	25	59.2	227	1127.9

Yices (version 2.4.1) does not support nested quantification, hence it was not applicable for the scholl class

Table 2 Results for LIA benchmarks, showing times (in seconds) and benchmarks solved by each solver and configuration over 4 benchmark classes with a 300s timeout

	psyco (189)		**sygus** (71)		**tptp** (46)		**uauto** (155)		**Total** (461)	
	#	Time	#	Time	#	Time	#	Time	#	Time
z3	189	10.7	71	7.0	46	2.1	155	6.1	461	25.9
cvc4	189	65.7	71	8.7	46	0.9	155	3.1	461	78.3
cvc4-sc16	189	26.6	67	268.9	46	1.4	155	3.7	457	300.7
z3-sc16	176	6.7	70	11.5	46	1.9	155	5.4	447	25.5
Vampire	26	145.7	59	522.0	44	3.2	155	54.4	284	725.4
Beagle	28	1330.4	55	378.9	46	54.1	153	389.2	282	2152.5
Princess	12	550.0	68	868.6	46	48.1	155	193.9	281	1660.6
VeriT	1	0.1	67	82.3	15	0.4	155	3.0	238	85.6
ProB	0	0.0	1	1.0	35	35.2	0	0.0	36	36.2

the techniques from [13], which we denote **yices** and **z3** respectively. For LRA, we consider 4 configurations of CVC4 each using different variants of the selection function in Fig. 2:

– **cvc4** uses the return value in Fig. 2,
– **cvc4+lw** uses the return value in Fig. 3 (Loos and Weispfenning),
– **cvc4+fr** uses the return value in Fig. 4 (Ferrante and Rackoff), and
– **cvc4+nvt** uses the return value in Fig. 5 (with no virtual terms).

By convention, all versions of CVC4 return instantiations for maximal lower bounds before minmial upper bounds, and do not use variable ordering heuristics for quantified formulas with multiple variables.

For both LRA and LIA, the best configuration of CVC4 solves the most benchmarks overall (616 and 461 respectively). Among the configurations of CVC4, the configuration using a selection function based on Loos and Weispfenning's method **cvc4+lw** performed the best. The performance of the latest version of Z3 has comparable performance, solving one fewer benchmark in LRA. The configuration **cvc4+lw** solves 5 benchmarks that **z3** does not. **z3** solves 4 benchmarks that **cvc4+lw** does not, and the virtual best of these solvers solves all but one benchmark within the timeout. Moreover, we note that **cvc4+lw** solves the aforementioned 5 benchmarks in an average of .4 s per benchmark. We believe that this is because CVC4's strategy is not restricted to quantified formulas that are in prenex normal form, and thus it may terminate quickly by realizing large nested disjunctions are not relevant to the overall formula. We will comment more on this in the next section. CVC4 solves more benchmarks (616) from the **scholl** class than any other solver due to its techniques for formulas with nested quantification from Sect 6. A technique [19] in the SMT solver Yices (version 2.4.1) is able to solve all benchmarks from the **keymaera** and **tptp** classes of LRA, but does not handle quantified formulas in LIA or with nested quantification.

For both benchmarks over LIA and LRA, the automated theorem provers trail the performance of CVC4 (and Z3) significantly. The best LRA automated theorem prover, Vampire, which uses a combination of a first-order theorem prover and an SMT solver [49], solves only 347 benchmarks, compared to 616 solved by **cvc4-fr**. The best LIA automated theorem prover was also Vampire, which solves 284 benchmarks, notably less than the 461 solved by CVC4. We conclude that recent lazy model-based techniques for quantified linear arithmetic, as implemented in CVC4 and Z3, are highly effective for solving quantified linear arithmetic.

8.1 Comparison of strategies for quantifier alternation on crafted benchmarks

In this subsection, we demonstrate the effectiveness of our approach for handling nested quantification based on the strategy in Sect. 6. As mentioned, our strategy may be applied to quantified formulas in a grammar that is not restricted to prenex normal form, and may often avoid reasoning about irrelevant portions of quantified formulas, as demonstrated in Example 11. This gives us an advantage with respect to existing approaches, including the strategy used in Z3 [13], which is limited to quantified formulas in prenex normal form.

To demonstrate this advantage, we constructed a set of benchmarks that are very easy if Boolean structure within quantifier scope is properly taken into account, and very hard otherwise. In detail, we randomly constructed 500 unique formula templates of the form $\varphi_1[F], \ldots, \varphi_{500}[F]$, where F occurs at some formula position in φ_i. Examples of formula templates of this form are $\bot \wedge F$, $\forall x\, \exists y\, (x > y \wedge F)$, $\exists xy\, x > y \wedge \exists z\, (x > z \wedge z > y) \wedge F$, and so on. These templates were constructed by recursively generating an abstract syntax tree where each formula node is given some probability of being one of $\forall$, $\exists$, $\neg$, $\wedge$, $\vee$, $>$, and each term node is given some probability of being either a bound variable or one of $+$, 0, 1. For each $i = 1, \ldots, 500$, we checked whether:

– The satisfiability result of $\varphi_i[\top]$ was the same as the satisfiability result of $\varphi_i[\bot]$, and
– The satisfiability of both $\varphi_i[\top]$ and $\varphi_i[\bot]$ could be quickly determined (in less than 5 s) by both CVC4 and Z3.

Among the 500 templates, we found that 360 met the first criterion above and 497 met the second criterion.[5] Overall, 357 met both criteria. Notice that for any such template that meets the first criteria, the satisfiability of $\varphi_i[F]$ can be determined independently of any closed formula we substitute for F. For each of these 357 templates, we measure the time taken by CVC4 and Z3 to solve the formula obtained by replacing F with closed formulas corresponding to hard benchmarks from the previous section. Since F is irrelevant to the satisfiability of

[5] For 2 templates, both Z3 and CVC4 took more than 5 s to solve for both cases of $F \in \{\top, \bot\}$, and for 1 template, Z3 timed out when F was $\bot$.

Table 3 Results for 357 randomly constructed templates of the form $\varphi[F]$ whose satisfiability does not depend on F, for five cases of F

Solver	F									
	$\top$		$\bot$		$F_{\text{rnd\_6\_39}}$		$F_{\text{rndpre\_4\_41}}$		$F_{\text{rndpre\_4\_56}}$	
	#	Time	#	Time	#	Time	#	Time	#	Time
cvc4	357	9.0	357	9.0	357	77.1	357	23.1	357	57.6
z3	357	8.0	357	7.9	297	17107.7	355	2043.9	356	2550.1

The first and second sets of columns give the number of solved and cumulative running time of **cvc4** and **z3** where F is $\top$ and $\bot$ respectively. The remaining three sets of columns give the results where F is formulas corresponding to hard benchmarks from the **scholl** class of SMT-LIB. All experiments run with a 300s timeout

$\varphi_i[F]$, one might expect that the satisfiability of $\varphi_i[F_{\text{hard}}]$ should be determined relatively quickly, even if the satisfiability of F_{hard} is hard to determine. However, we have that this is not necessarily the case, and that solving time can vary significantly from solver to solver.

Table 3 give the results of solvers **cvc4** and **z3** on template and instantiation pairs. We consider the 357 formula templates, as described above. The first two sets of columns give CVC4 and Z3's cumulative run time on the 357 templates where F is replaced by $\top$ and $\bot$ respectively. For the remaining three sets of columns, we replace F by formulas corresponding to benchmarks from the **scholl** class of SMT-LIB (rnd_6_39, rndpre_4_41, and rndpre_4_56). We chose these benchmarks since they are the three benchmarks that neither CVC4 nor Z3 can solve within a two minute timeout. We found that no solver answered differently for two queries of the form $\varphi[F_1]$ and $\varphi[F_2]$ for $F_1 \neq F_2$. In this experiment, Z3 timed out for 169 templates instantiated with $F_{\text{rnd\_6\_39}}$, and 2 templates instantiated with $F_{\text{rndpre\_4\_41}}$. Overall, Z3 solved 71.2% benchmarks from the last three sets columns in less than 30 s. On the other hand, the results show that CVC4 solves each benchmark relatively quickly, regardless of the contents of F. The longest it took to solve any benchmark was 4.6 s. When comparing the average solving time for benchmarks in the last three sets of columns versus the first two sets, the average overhead was .13 s per benchmark.

For a closer look, Table 4 gives results for an additional five crafted templates, and the times taken by CVC4 and Z3 to solve each (template, benchmark) pair. For all templates, we assume when applicable that free constants (e.g. a and b) are existentially quantified. The first four templates are unsatisfiable regardless of the content of F. The fifth template we considered was of the form $F_{\text{rndpre\_4\_8}} \vee F$, where $F_{\text{rndpre\_4\_8}}$ is the formula corresponding to a benchmark (also from the **scholl** class) which CVC4 and Z3 both find to be satisfiable quickly, and thus this template is satisfiable for all F. For the first template, both CVC4 and Z3 solve all three instances quickly. For the other templates, Z3's performance varies significantly. In fact, for some benchmarks such as the fourth template where F is $F_{\text{rnd\_6\_39}}$, Z3's performance is worse (>300 s) than running on $F_{\text{rnd\_6\_39}}$ alone, which it solves in 226.3 s. Overall, CVC4 answers quickly for all benchmarks, solving almost all benchmarks in less than a second. It takes around 4 s to solve the second and third templates instantiated with rndpre_4_56, where it spends a majority of its time finding a model for the initial set of quantifier-free constraints and terminates after only two iterations of the loop in procedure CEGQI_T from Fig. 8. In our testing, we were unable to find a template $\varphi[F]$ for which CVC4 took more than 5 s longer to solve $\varphi[F_{\text{hard}}]$ when compared to the time it took to solve $\varphi[\top]$, where F_{hard} is a formula corresponding to one of the benchmarks in the columns of Table 4. On the other hand, Z3's performance varied significantly on many of the non-trivial templates we considered.

Table 4 Results for example crafted templates $\varphi[F]$ for three cases of F. The formula $F_{\text{rndpre\_4\_8}}$ corresponds to an easy satisfiable benchmark from the same class

Template	$F_{\text{rnd\_6\_39}}$		$F_{\text{rndpre\_4\_41}}$		$F_{\text{rndpre\_4\_56}}$	
	Solver	Time	Solver	Time	Solver	Time
$a > b \wedge \exists y\, (a > y \wedge y > b) \wedge F$	CVC4	0.17	CVC4	0.05	CVC4	0.21
	Z3	0.13	Z3	0.03	Z3	0.06
$\forall x\, \exists y\, (\forall w\, \exists z\, (w > z \wedge x > y \wedge y > a) \wedge \neg F)$	CVC4	0.28	CVC4	0.19	CVC4	4.15
	Z3	203.5	Z3	209.7	Z3	29.17
$\forall x\, \exists y\, \forall w\, \exists z\, (w > z \wedge x > y \wedge y > a \wedge \neg F)$	CVC4	0.28	CVC4	0.19	CVC4	4.22
	Z3	234.0	Z3	172.9	Z3	75.5
$\forall x\, \exists y\, ((\forall w\, \exists z\, \forall u\, w > z \wedge u > z \wedge z > w \wedge w > y \wedge y > x) \wedge F)$	CVC4	0.16	CVC4	0.04	CVC4	0.09
	Z3	>300	Z3	0.69	Z3	3.2
$F_{\text{rndpre\_4\_8}} \vee F$	CVC4	0.16	CVC4	0.04	CVC4	0.10
	Z3	196.4	Z3	1.85	Z3	8.9

We believe this indicates that a strategy for quantified linear arithmetic that does not require formulas to be in prenex normal form, such as the one from Sect. 6, can have significant performance advantages. We conjecture that this design decision accounts for the differences between CVC4 and Z3 in our evaluation in the previous section, where we found that for 5 of the original benchmarks from the **scholl** class for which Z3 timed out, at least one configuration of CVC4 was able to solve in less than 2 s.

8.2 Comparison with synthesis solvers

The techniques for solving quantified linear arithmetic in CVC4 have the additional advantage that they may produce solutions to synthesis conjectures as described in Sect. 7. In the 2015 edition of the syntax-guided synthesis (SyGuS) competition [2], an earlier version of CVC4 won the conditional linear arithmetic track, solving 70 of 73 benchmarks using techniques from [50]. The nearest solver ALCHEMIST [53] solved 43. CVC4 also won the conditional linear arithmetic track in the 2016 edition, solving all 73 synthesis conjectures, and a new approach in EUsolver [3] solved 72. The improvement in CVC4 in the latest edition is due to its use of a complete instantiation strategy for linear integer arithmetic when solving single invocation synthesis conjectures as described in Sect. 7.

8.3 Number of instantiations

We measured statistics on the number of instantiations CVC4 constructs while solving benchmarks in the previous section. Let $\#\mathsf{lits}(\psi, X)$ be the number of literals in ψ containing a variable in set X. We approximate the number of possible instantiations of a quantified formula ψ of the form $\forall x_1 \ldots x_n\, \varphi$ where φ is quantifier-free by the following calculation:

$$\#\mathsf{PInst}(\psi) = \Pi_{i=1,\ldots,n}\mathsf{max}(1, \#\mathsf{lits}(\varphi, \{x_i\})$$

Table 5 Average number of instantiations and possible instantiations per benchmark for LRA and LIA benchmarks

	keymaera (222)		**scholl** (351)		**tptp** (25)	
	#Inst	#PInst	#Inst	#PInst	#Inst	#PInst
cvc4	0.11	0.86	7.23	5937.8	0.04	0.08

	psyco (189)		**sygus** (71)		**tptp** (46)		**uauto** (155)	
	#Inst	#PInst	#Inst	#PInst	#Inst	#PInst	#Inst	#PInst
cvc4	21.9	>1M	5.1	7.6	0.2	0.3	1.6	6.3

For each i, we add a factor corresponding to an approximation of the number of possible bounds for variable x_i. This measure is proportional to the worst-case behavior of the total number of instantiations required for the termination of the instantiation-based procedure in Fig. 1 when using the selection functions $\mathcal{S}_{\text{LRA}}$ and $\mathcal{S}_{\text{LIA}}$ for linear real and integer arithmetic. More precisely, each factor for variable x_i is an approximation of the size of the set of possible terms returned by functions S_{R0} and S_{I0}, as described in the proofs of Lemmas 3 and 6 respectively.

Table 5 gives the average number of instantiations considered by cvc4 and the average possible number of instantiations considered by cvc4 across all benchmark families on benchmarks that all configurations of cvc4 solve. A few benchmark families (such as **tptp** and **keymaera**) had a very small number of possible instantiations, which can partially be attributed to the fact that cvc4 applies aggressive preprocessing techniques to eliminate variables from quantified formulas. Conversely, other benchmark families (such as **scholl** and **psyco**) had a very large number of possible instantiations. Many of the benchmarks in **psyco** contained formulas with quantifier prefixes up to 50 variables in length, and hence often had upwards of 2^{50} possible instantiations. Since the benchmarks in **scholl** contain nested quantification, we consider a very conservative estimate of the number of possible instantiations by only considering innermost quantified formulas in our computation. The number of instantiations cvc4 considers is on average considerably less the number of possible instantiations, demonstrating that a lazy approach for quantifier instantiation is beneficial, and often critical, to solving benchmarks in these libraries.

9 Conclusion

We have presented a class of instantiation-based procedures that are at the same time complete for quantified linear arithmetic and highly efficient in practice. Thanks to our framework we also obtain a simple and modular correctness argument for soundness and completeness on formulas with one quantifier alternation. This correctness argument is used in part for showing soundness and completeness on formulas with arbitrary quantifier alternations, as well as a complete and efficient method for solving single invocation synthesis conjectures. Our procedure for arbitrary quantifier alternations has advantages over approaches that are limited to formulas in prenex normal form.

For future work, we would like to extend the approach to new theories including fixed-width bit vectors, strings, and non-linear arithmetic, as well as for combinations of theories that admit quantifier elimination. We would like to focus on further heuristics for quantified linear arithmetic with arbitrary quantifier alternations, and for avoiding worst case performance for quantified integer arithmetic involving large coefficients. A longer term goal of this work is to develop an approach that is effective in practice for quantified formulas involving both background theories and uninterpreted functions. We plan to investigate the use of the framework described in this paper as a component of such an approach.

Acknowledgements We would like to thank Mikolas Janota for his helpful discussion, and Peter Baumgartner for his help with converting the benchmarks used in the evaluation to the TPTP format.

References

1. Alur R, Bodik R, Dallal E, Fisman D, Garg P, Juniwal G, Kress-Gazit H, Madhusudan P, Martin MMK, Raghothaman M, Saha S, Seshia SA, Singh R, Solar-Lezama A, Torlak E, Udupa A (2014) Syntax-guided synthesis. To Appear in Marktoberdorf NATO proceedings
2. Alur R, Fisman D, Singh R, Solar-Lezama A (2016) Results and analysis of sygus-comp'15. arXiv preprint arXiv:1602.01170
3. Alur R, Radhakrishna A, Udupa A (2016) Scaling enumerative program synthesis via divide and conquer. Technical report, UPenn https://www.seas.upenn.edu/~arjunrad/publications/eusolver_report.pdf
4. Backofen R (1995) A complete axiomatization of a theory with feature and arity constraints. J Log Program 24:37–72
5. Bansal K, Reynolds A, King T, Barrett C, Wies T (2015) Deciding local theory extensions via e-matching. In: Computer aided verification (CAV), Springer
6. Barrett C, Conway C, Deters M, Hadarean L, Jovanovic D, King T, Reynolds A, Tinelli C (2011) Cvx4. In: Computer aided verification (CAV), Springer
7. Barrett C, Stump A, Tinelli C (2010) The satisfiability modulo theories library (SMT-LIB). http://www.SMT-LIB.org
8. Baumgartner P (2015) Smttotptp a converter for theorem proving formats. In: CADE-25, Lecture notes in computer science, vol 9195. Springer
9. Berman L (1980) The complexity of logical theories. Theor Comput Sci 11(1):71–77
10. Beyene TA, Chaudhuri S, Popeea C, Rybalchenko A (2014) A constraint-based approach to solving games on infinite graphs. In: POPL, pp 221–234
11. Beyene TA, Popeea C, Rybalchenko A (2013) Solving existentially quantified Horn clauses. In: CAV, pp 869–882
12. Bjørner N (2010) Linear quantifier elimination as an abstract decision procedure. In Giesl J, Hähnle R (eds) IJCAR, LNCS, vol 6173. Springer, pp 316–330
13. Bjørner N, Janota M (2015) Playing with quantified satisfaction. In: 20th international conferences on logic for programming, artificial intelligence and reasoning—short presentations, LPAR 2015, Suva, Fiji, 24–28 November 2015, pp 15–27
14. Bjørner N, McMillan KL, Rybalchenko A (2012) Program verification as satisfiability modulo theories. In: SMT@IJCAR, pp 3–11
15. Comon H, Delor C (1994) Equational formulae with membership constraints. Inf Comput 112(2):167–216
16. Cooper DC (1972) Theorem proving in arithmetic without multiplication. In: Meltzer B, Michie D (eds) Machine intelligence, vol 7. Edinburgh University Press, Edinburgh, pp 91–100
17. de Moura LM, Bjørner N (2007) Efficient e-matching for SMT solvers. In: Pfenning F, (ed) CADE, LNCS, vol 4603. Springer, pp 183–198
18. Detlefs D, Nelson G, Saxe JB (2003) Simplify: a theorem prover for program checking. J. ACM, Technical report
19. Dutertre B (2015) Solving exists/forall problems with yices. In: Workshop on Satisfiability modulo theories
20. Farzan A Kincaid Z (2016) Linear arithmetic satisfiability via strategy improvement. In: Proceedings of the twenty-fifth international joint conference on artificial intelligence, IJCAI 2016, New York, NY, USA, 9–15 July 2016, pp 735–743
21. Fedyukovich G, Gurfinkel A, Sharygina N (2015) Automated discovery of simulation between programs. In: Logic for programming, artificial intelligence, and reasoning—20th international conference, LPAR-20 2015, Suva, Fiji, 24–28 November 2015, Proceedings, pp 606–621

22. Feferman S, Vaught RL (1959) The first order properties of products of algebraic systems. Fundam Math 47:57–103
23. Ferrante J, Rackoff CW (1979) The computational complexity of logical theories, lecture notes in mathematics, vol 718. Springer, Berlin
24. Ganzinger H, Korovin K (2003) New directions in instantiation-based theorem proving. In: Logic in computer science, 2003. IEEE
25. Ge Y, Barrett C, Tinelli C (2007) Solving quantified verification conditions using satisfiability modulo theories. In CADE, LNCS, vol 4603. Springer
26. Ge Y, de Moura L (2009) Complete instantiation for quantified formulas in satisfiability modulo theories. In: Proceedings of CAV'09, LNCS, vol 5643. Springer
27. Grebenshchikov S, Lopes NP, Popeea C, Rybalchenko A (2012) Synthesizing software verifiers from proof rules. In: PLDI, pp 405–416
28. Heizmann M, Dietsch D, Leike J, Musa B, Podelski A (2015) Ultimate automizer with array interpolation. In: TACAS
29. Hodges W (1993) Model Theory, encyclopedia of mathematics and its applications, vol 42. Cambridge University Press, Cambridge
30. Jacobs S (2009) Incremental instance generation in local reasoning. In: CAV '09, Springer, Berlin, Heidelberg, pp 368–382
31. Janota M, Klieber W, Marques-Silva J, Clarke E (2012) Solving qbf with counterexample guided refinement. In: International conference on theory and applications of satisfiability testing, Springer, Berlin, Heidelberg, pp 114–128
32. Komuravelli A, Gurfinkel A, Chaki S (2014) SMT-based model checking for recursive programs. In: Computer aided verification, Springer
33. Kozen D (2006) Theory of computation. Springer, Berlin
34. Kuncak V, Mayer M, Piskac R, Suter P (2010) Complete functional synthesis. In: Zorn BG, Aiken A (eds) PLDI. ACM, New york, pp 316–329
35. Kuncak V, Rinard M (2003) Structural subtyping of non-recursive types is decidable. In: Eighteenth annual IEEE symposium on logic in computer science (LICS). IEEE
36. Loos R, Weispfenning V (1993) Applying linear quantifier elimination. Comput J 36(5):450–462
37. Lopes NP, Monteiro J (2014) Weakest precondition synthesis for compiler optimizations. In: VMCAI 2014, pp 203–221
38. Maher MJ (1988) Complete axiomatizations of the algebras of the finite, rational, and infinite trees. In: IEEE symposium on logic in computer science
39. Mal'cev AI (1971) The metamathematics of algebraic systems, studies in logic and the foundations of mathematics, vol 66. North-Holland, Amsterdam
40. Monniaux D (2009) Automatic modular abstractions for linear constraints. In: POPL 2009, pp 140–151
41. Monniaux D (2010) Quantifier elimination by lazy model enumeration. In: Touili T, Cook B, Jackson P, (eds), CAV, LNCS, vol 6174. Springer, pp 585–599
42. Mostowski A (1952) On direct products of theories. J Symb Logic 17(1):1–31
43. Nipkow T (2008) Linear quantifier elimination. In: Automated reasoning, pp 18–33
44. Phan A, Bjørner N, Monniaux D (2012) Anatomy of alternating quantifier satisfiability (work in progress). In SMT 2012
45. Platzer A, Quesel J-D, Rümmer P (2009) Real world verification. In: Automated Deduction–CADE-22, Springer, Berlin, Heidelberg, pp 485–501
46. Presburger M (1929) über die vollständigkeit eines gewissen systems der aritmethik ganzer zahlen, in welchem die addition als einzige operation hervortritt. In: Comptes Rendus du premier Congrès des Mathématiciens des Pays slaves, Warsawa, pp 92–101
47. Pugh W (1991) The Omega test: a fast and practical integer programming algorithm for dependence analysis. In: ACM/IEEE conference supercomputing
48. Reddy CR, Loveland DW (1978) Presburger arithmetic with bounded quantifier alternation. In: ACM STOC, ACM Press, pp 320–325
49. Reger G, Suda M, Voronkov A (2015) Playing with avatar. In: Automated deduction-CADE-25, Springer, pp 399–415
50. Reynolds A, Deters M, Kuncak V, Tinelli C, Barrett CW (2015) Counterexample-guided quantifier instantiation for synthesis in SMT. In: Computer aided verification —27th international conference, CAV 2015, San Francisco, CA, USA, 18-24 July 2015, Proceedings, Part II, pp 198–216
51. Reynolds A, Tinelli C, Moura LD (2014) Finding conflicting instances of quantified formulas in SMT. In: Formal methods in computer-aided design (FMCAD)
52. Rybina T, Voronkov A (2001) A decision procedure for term algebras with queues. ACM Trans Comput Logic (TOCL) 2(2):155–181

53. Saha S, Garg P, Madhusudan P (2015) Alchemist: learning guarded affine functions. In: Computer aided verification—27th international conference, CAV 2015, San Francisco, CA, USA, 18–24 July 2015, Proceedings, Part I, pp 440–446
54. Scholl C, Disch S, Pigorsch F, Kupferschmid S (2008) Using an smt solver and craig interpolation to detect and remove redundant linear constraints in representations of non-convex polyhedra. In: SMT, ACM, pp 18–26
55. Skolem T (1919) Untersuchungen über die Axiome des Klassenkalküls and über "Produktations- und Summationsprobleme", welche gewisse Klassen von Aussagen betreffen. Skrifter utgit av Vidnskapsselskapet i Kristiania, I. klasse, no. 3, Oslo
56. Sturm T, Tiwari A (2011) Verification and synthesis using real quantifier elimination. In: ISSAC 2011, pp 329–336
57. Sturm T, Weispfenning V (2002) Quantifier elimination in term algebras: the case of finite languages. TUM Muenchen, In: Computer algebra in scientific computing (CASC)
58. Sutcliffe G (2009) The TPTP problem library and associated infrastructure: the FOF and CNF parts, v3.5.0. J Autom Reason 43(4):337–362
59. Sutcliffe G (2016) The CADE ATP system competition—CASC. AI Magazine 37(2):99–101
60. Tarski A (1949) Arithmetical classes and types of algebraically closed and real-closed fields. Bull Am Math Soc 55(1):64
61. Tarski A (1949) Arithmetical classes and types of boolean algebras. Bull Am Math Soc 55(64):1192
62. Treinen R (1997) Feature trees over arbitrary structures, chapter 7. In: Blackburn P, de Rijke M (eds) Specifying syntactic structures. CSLI Publications and FoLLI, Stanford
63. Walukiewicz I (2002) Monadic second-order logic on tree-like structures. Theor Comput Sci 275(1–2):311–346
64. Weispfenning V (1997) Complexity and uniformity of elimination in Presburger arithmetic. In: ISSAC '97, New York, NY, USA, ACM, pp 48–53
65. Weispfenning V (1999) Mixed real-integer linear quantifier elimination. In: Proceedings of the 1999 international symposium on symbolic and algebraic computation, ISSAC '99, New York, NY, USA, ACM, pp 129–136
66. Wintersteiger CM, Hamadi Y, De Moura L (2013) Efficiently solving quantified bit-vector formulas. Form Methods Syst Des 42(1):3–23