An Instantiation-Based Approach for Solving Quantified Linear Arithmetic

Andrew Reynolds\(^1\), Tim King\(^2\), and Viktor Kuncak\(^1\)

\(^1\) École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
\(^2\) Verimag

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 (LRA) and linear integer arithmetic (LIA) formulas with one quantifier alternation. Our procedure can be integrated into the solving architecture used by typical SMT solvers. Experimental results on standardized benchmarks from model checking, static analysis, and synthesis show that our implementation of the procedure in the SMT solver CVC\(^4\) outperforms existing tools for quantified linear arithmetic.

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 [31,47,53], construction of transfer functions for program analysis [37], invariant inference [13,25], as well as analysis of properties that go beyond safety [9,10].

The most commonly used complete method for deciding constraints over quantified theories is quantifier elimination [27, 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 [8], 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 [61]. 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 [11,38] 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 [17], which are incomplete but work well in practice for undecidable fragments of first-order logic.

Thus, our motivation is to capitalize both on recent advances in specialized techniques for quantified linear arithmetic [11, 12, 29, 41], and recent advances in instantiation-based theorem proving for first-order logic [16, 22, 49]. This paper seeks to bridge the gap between these two lines of research by introducing 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 [16, 17, 49], 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 [47], 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 [52, 58], Presburger arithmetic [43], decidability of products [20, 39], [36, Chapter 12], and algebraically closed fields [57]. The original result on decidability of Presburger arithmetic is by Presburger [43]. The space bound for Presburger arithmetic was shown in [21]. The matching lower and upper bounds for Presburger arithmetic were shown in [8], see also [30, Lecture 24]. An analysis parameterized by the number of quantifier alternations is presented in [45]. A mechanically verified quantifier elimination algorithm was developed by Nipkow [40].

An approach for lazy quantifier elimination for linear real arithmetic was developed by Monniaux [38]. Integration of linear quantifier elimination into the solving algorithm used by SMT solvers was developed in [11], 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 [29], and can be used for extracting skolem functions for simulation synthesis [19]. A recent approach for quantified formulas with arbitrary alternations has been developed by Bjorner [12] for several background theories, which is not based on instantiation. The most widely used techniques for quantifier instantiation in SMT were developed in [17], and later in [16, 23], which primarily focused on 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 to decide essentially uninterpreted fragment [24], and, more generally, theories having a locality property [3, 28]; 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 [18], which does not treat linear integer arithmetic. The present paper builds upon our previous work for solving synthesis conjectures using quantifier instantiation in
SMT [47], 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 [36, Chapter 23], [35, 54] and their extensions [14, 32, 50], feature trees [2, 59], and monadic second-order theories [60].

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 Section 2, and show that these procedures can be used in part as an approach for solving synthesis problems in Section 2.3. 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 Sections 3 and 4, two quantified fragments for which many current SMT solvers do not have efficient support for. We show how our procedure can be integrated into the solving architecture used by SMT solvers in Section 5. To our knowledge, our approach is the first complete algorithm for quantified linear arithmetic with one alternation that is based purely on quantifier instantiation, which has the advantage of being composable with existing techniques and whose soundness is straightforward to verify. In Section 6, we demonstrate 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 $\Sigma$ consists of a countable set of sort symbols and a set of function symbols. Given a signature $\Sigma$, well-sorted terms, atoms, literals, and formulas are defined as usual, and referred to respectively as $\Sigma$-terms. We denote by $FV(t)$ the set of free variables occurring in the term $t$, and extend this notion to formulas. A $\Sigma$-term or formula is ground if it has no free variables. A term written $t[k]$ denotes a term whose free variables are in $k$.

A $\Sigma$-interpretation $I$ maps

- each sort symbol $\sigma \in \Sigma$ to a non-empty set $\sigma^I$, the domain of $\sigma$ in $I$, and
- each function $f \in \Sigma$ of sort $\sigma_1 \times \ldots \times \sigma_n \rightarrow \sigma$ to a total function $f^I$ of sort $\sigma_1^I \times \ldots \times \sigma_n^I \rightarrow \sigma^I$ where $n > 0$, and to an element of $\sigma^I$ when $n = 0$.

We write $I^\Sigma$ to denote the interpretation of $t$ in $I$, defined inductively as usual. A satisfiability relation between $\Sigma$-interpretations and $\Sigma$-formulas, written $I \models \varphi$, is also defined inductively as usual. In particular, we assume that $I \models \neg \varphi$ if and only if it is not the case that $I \models \varphi$. We say that $I$ is a model of $\varphi$ if $I$ satisfies $\varphi$. Formulas $\varphi_1$ and $\varphi_2$ are equivalent (up to $k$) if they are satisfied by the same set of models (when restricted to the interpretation of variables $k$).

A theory is a pair $T = (\Sigma, I)$ where $\Sigma$ is a signature and $I$ is a non-empty set of $\Sigma$-interpretations, the models of $T$. We assume $\Sigma$ contains the equality predicate, which we denote by $\approx$. Let $[\varphi]_T$ denote the set of $T$-models of $\varphi$. Observe that $[\neg \varphi]_T = \{ I \in T : I \models \neg \varphi \}$. 


I \ \models [\varphi]_T, A \Sigma\text{-formula } \varphi[x] \text{ is } T\text{-satisfiable if it is satisfied by some interpretation in } I \text{ (i.e. } [\varphi]_T \neq \emptyset\text{). Dually, a } \Sigma\text{-formula } \varphi[x] \text{ is } T\text{-unsatisfiable if it is satisfied by no interpretation in } I \text{ (i.e. } [\varphi]_T = \emptyset\text{). A formula } \varphi \text{ is } T\text{-valid if every model of } T \text{ is a model of } \varphi \text{ (i.e., } [\varphi]_T = I\text{). Given a fragment } \mathcal{L} \text{ of the language of } \Sigma\text{-formulas, a } \Sigma\text{-theory } T \text{ is satisfaction complete with respect to } \mathcal{L} \text{ if every closed } T\text{-satisfiable formula of } \mathcal{L} \text{ is } T\text{-valid. In terms of set of models, satisfaction completeness means that } [\varphi]_T \neq \emptyset \text{ implies } [\varphi]_T = I, \text{ or, in other words, for every } F \subseteq \mathcal{L} \text{ exactly one of the following two cases hold: } [\varphi]_T = \emptyset, \text{ or } [\varphi]_T = I. \text{ If additionally } \mathcal{L} \text{ is closed under negation, then, for every } \varphi \in \mathcal{L}, \text{ either } \varphi \text{ or } \neg F \text{ is unsatisfiable.}

A set } \Gamma \text{ of formulas } T\text{-entails a } \Sigma\text{-formula } \varphi, \text{ written } \Gamma \models_T \varphi, \text{ if every model of } T \text{ that satisfies all formulas in } \Gamma \text{ satisfies } \varphi \text{ as well. A set of literals } M \text{ propositionally entails a formula } \varphi, \text{ written } M \models_p \varphi, \text{ if } M \text{ entails } \varphi \text{ when considering all atomic formulas in } M \cup \varphi \text{ as propositional variables; such entailment is one of from propositional logic and is independent of the theory. We write } RA (\text{resp. } IA) \text{ to denote the theory of real (resp. integer) arithmetic. Its signature consists of the sort } \text{Real} (\text{resp. } \text{Int}), \text{ the binary predicate symbols } > \text{ and } <, \text{ functions } + \text{ and } \cdot \text{ denoting addition and multiplication, and the constants of its sort interpreted as usual. We write } t \leq s \text{ as shorthand for } \neg(t > s), \text{ and } t \geq s \text{ as shorthand for } \neg(t < s). \text{ We write } LRA \text{ (resp. } LIA) \text{ 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 + \ldots + c_n \cdot x_n > c) \text{ where } c_1, \ldots, c_n, c \text{ and } x_1, \ldots, x_n \text{ are non-zero constants and distinct variables of sort } \text{Real} \text{ (resp. } \text{Int}) \text{ respectively, and } > \text{ is one of } >, <, \text{ or } =. \text{ For each literal of this form, there exists an equivalent literal that is in solved form with respect to } x_i \text{ for each } i = 1, \ldots, n. \text{ That is, an } LRA\text{-literal is in solved form with respect to } x \text{ if it is of the form } (\neg)(x > t), \text{ where } x \not\in FV(t). \text{ Similarly, an } LIA\text{-literal is in solved form with respect to } x \text{ if it is of the form } (\neg)(c \cdot x > t), \text{ where } x \not\in FV(t) \text{ and } c \text{ is an integer constant greater than zero. For integer constants } c_1 \text{ and } c_2 \text{ and non-zero constant } c, \text{ we write } c_1 \equiv_c c_2 \text{ to denote that } c_1 \text{ and } c_2 \text{ are congruent modulo } c, \text{ that is } (c_1 \text{ mod } c) = (c_2 \text{ mod } c), \text{ and we write } c \mid c_1 \text{ if } c \text{ divides } c_1.

2 \text{ Quantifier Instantiation for Theories}

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

2.1 \text{ An Instantiation Procedure and Its Soundness}

Figure 1 presents an instantiation-based approach for determining the satisfiability of a } T\text{-formulas } \exists a \forall x \varphi[a, x], \text{ where } \varphi[a, x] \text{ belongs to } \mathcal{L}. \text{ The procedure introduces a tuple of distinct fresh constants } k \text{ of the same sort as } a, \text{ and } e \text{ of the same sort as } x. \text{ It maintains a set of formulas } \Gamma, \text{ initially empty, and terminates when either } \Gamma \text{ or } \Gamma \cup \{ \neg \varphi[k, e] \} \text{ is } T\text{-unsatisfiable. On each iteration, the procedure invokes the subprocedure } S \text{ (over which the procedure is parameterized), which returns a tuple of terms}
Lemma 2

If $\vdash \exists a \forall x. \varphi[a, x]$ is unsatisfiable, then return "unsat".

Proof: Let $I$ be a model of $\exists a \forall x. \varphi[a, x]$ such that $\varphi[a, t]$ is $I$-unsatisfiable, then return "unsat".

Otherwise, let $I$ be a model of $\exists a \forall x. \varphi[a, x]$, and let $I[a] = S(I, \varphi[a, e])$. $I[a]$ is also in $\vdash \exists a \forall x. \varphi[a, x]$.

Repeat

- If $I'$ is $T$-unsatisfiable, then return "unsat".
- Otherwise, let $I'$ be a model of $T$ and $I'$ and let $t[k] = S(I, \varphi[k, e])$. $I'$ is also in $\vdash \exists a \forall x. \varphi[a, x]$. $I'$ is also in $\vdash \exists a \forall x. \varphi[a, x]$.

Fig. 1. An instantiation-based approach $\mathcal{P}_S$ for determining the $T$-satisfiability of $\exists a \forall x. \varphi[a, x]$ parametrized by selection function $S$. $t[k]$ whose free variables are a subset of $k$. We then add to $I$ the formula $\varphi[k, t[k]] \downarrow$, a formula equivalent to $\varphi[k, t[k]]$ up to $k^3$. We call $S$ the selection function of $\mathcal{P}_S$.

The intuition of the algorithm is to find a subset of the instances of $\exists a \forall x. \varphi[a, x]$ that are either (a) unsatisfiable, and are thus sufficient for showing that $\exists a \forall x. \varphi[a, x]$ is unsatisfiable, or (b) satisfiable and entail $\exists a \forall x. \varphi[a, x]$. The algorithm recognizes the latter case by checking the satisfiability of $\exists a \forall x. \varphi[a, e]$ on each iteration of its main loop. In either case, the algorithm may terminate before enumerating all instances of $\exists a \forall x. \varphi[a, x]$. In practice, we have found that the algorithm often terminates after enumerating only a small number of instances for benchmarks that occur in practice.

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

Note that a selection function is only defined if $I$ is a model for $\exists a \forall x. \varphi[a, e]$. 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. Detailed proofs of the all claims in this paper can be found in the extended version of this report [48].

Lemma 1 If $\mathcal{P}_S$ terminates with "unsat", then $\exists a \forall x. \varphi[a, x]$ is $T$-unsatisfiable.

Proof: In this case, there exists a set $\Gamma$ that is equivalent to $\{\varphi[k, t_1], \ldots, \varphi[k, t_p]\}$ and is $T$-unsatisfiable where $k$ are distinct fresh constants. Thus, $\exists a \forall x. \varphi[k, x]$ is $T$-unsatisfiable. Since $k$ are distinct and fresh, we conclude that $\exists a \forall x. \varphi[a, x]$ is $T$-unsatisfiable.

Lemma 2 If $\mathcal{P}_S$ terminates with "sat", then $\exists a \forall x. \varphi[a, x]$ is $T$-satisfiable.

Proof: In this case, there exists a set $\Gamma'$ equivalent to $\{\varphi[k, t_1], \ldots, \varphi[k, t_p]\}$ that is $T$-satisfiable, where $k$ are distinct fresh constants, and $\Gamma'$ equivalent to $\Gamma \cup \{\neg \varphi[k, e]\}$ that is $T$-unsatisfiable. The variables $e$ do not occur in $\Gamma$, since the only formulas added to it are of the form $\varphi[k, t[k]] \downarrow$. Thus, we have that $\Gamma' \cup \{\exists x. \neg \varphi[k, x]\}$ is $T$-unsatisfiable. Let $I$ be a model of $\Gamma$. Since $I$ is not a model of $\Gamma'$, it must be the case that $I \not\models \exists x. \neg \varphi[k, x]$, and hence $I$ is a model for $\exists a \forall x. \varphi[a, x]$.

3 We further comment on examples of operators $\downarrow$ in Sections 3 and 4.
The connection between quantifier elimination and synthesis has been shown fruitful in previous work [31]; it is one of our motivations for further improving quantified reasoning modulo theories. The procedure mentioned in this section can be used to synthesize functions from certain classes of specifications. Consider (second-order) $T$-formulas of the form:

$$
\exists f \forall x \varphi[f, x]
$$

of the form $\exists f \forall x \varphi[f, x]$, where $\varphi$ is a quantifier-free formula, $x = (x_1, \ldots, x_n)$ is a tuple of variables of sort $\sigma$, for $i = 1, \ldots, n$, and $f = (f_1, \ldots, f_m)$ is a tuple of
functions of sort \(\sigma_1 \times \ldots \times \sigma_n \rightarrow \tau_j\) for \(j = 1, \ldots, m\). We call such formulas synthesis conjectures. A synthesis conjecture is single invocation (over \(L\)) if it is equivalent to:

\[
\exists f \forall x \psi[x, f(x)] 
\]  

(2)

where \(\psi[x, y] \in L\). That is, functions from \(f\) are applied to the tuple \(x\) only. The formula (2) is equivalent to the (first-order) formula \(\forall x \exists y \psi[x, y]\), whose negation

\[
\exists x \forall y \neg \psi[x, y] 
\]  

(3)

is suitable as an input to Figure 1. As observed in [47], solutions for single invocation synthesis conjectures can be extracted from an unsatisfiable core of instantiations when proving the unsatisfiability of (3). In particular, let \(k\) be a set of distinct fresh variables of the same sort as \(x\), and say the set \(\{ \neg \psi[k, t_1[k]], \ldots, \neg \psi[k, t_p[k]] \}\) is \(T\)-unsatisfiable where \(t_i = (t_{i1}[k], \ldots, t_{in}[k])\) for \(i = 1, \ldots, p\). Then:

\[
1 \leq j \leq m : f_j = \lambda x. [\text{ite}(\psi[x, t_{j1}[x]], t_{j1}[x], \ldots \text{ite}(\psi[x, t_{jn}[x]], t_{jn}[x], t_{jn}[x]))] \quad (4)
\]

is a solution for \(f\) in (2). We use the instantiation-based procedure in Figure 1 for discharging (3). In contrast to prior work [47], we here devise selection functions \(S\) for \(L\) that are finite and monotonic, obtaining a sound and complete method for synthesizing tuples of functions whose specification is a single invocation synthesis conjecture over \(L\). In the following sections, we show such selection functions both for linear real arithmetic (LRA) and linear real arithmetic (LIA).

It is important to note that the solution (4) does not necessarily belong to the language \(L\), since there is no restriction on the selection functions for \(L\) that restricts its return value \(t\) to terms in \(L\). For example, in our approach for linear real arithmetic, \(t\) may contain a free distinguished constant \(\delta\) representing an infinitesimal positive value, and in our approach for linear integer arithmetic, \(t\) may contain integer division. Additional steps may be necessary in practice for making (4) a computable function.

### 2.4 Illustration: Instantiation for a Simple Fragment of LRA

We first present a selection function for a restricted class \(L\) of LRA-formulas \(\exists a \forall x. \varphi[a, x]\), namely whose universal quantifier is over single variable \(x\) of sort Real, \(a\) are variables of sort Real, and whose (skolemized) body \(\varphi[k, e]\) is of the form:

\[
(e < \ell_1 \lor \ldots \lor e < \ell_n \lor e > u_1 \lor \ldots \lor e > u_m) 
\]  

(5)

where at least one of \(\{n, m\}\) is greater than zero, and \(e \notin \text{FV}(\ell_1, \ldots, \ell_n, u_1, \ldots, u_m)\). Figure 2 gives a selection function for \(S_{\text{SLRA}}\). It considers the interpretation of terms \(\ell_1, \ldots, \ell_n\) and \(u_1, \ldots, u_m\) in a model \(I\) of \(\Gamma\). If \(n > 0\), then \(S_{\text{SLRA}}\) returns the \(\ell_j\) whose value is maximal in \(I\). If \(n = 0\), then \(m > 0\) and \(S_{\text{SLRA}}\) returns the \(u_i\) whose value is minimal in \(I\).

**Lemma 4**: \(S_{\text{SLRA}}\) is finite for \(\varphi[k, e]\).

**Proof**: The terms returned by \(S_{\text{SLRA}}\) are in \(\{\ell_1, \ldots, \ell_n\}\) when \(n > 0\), and in \(\{u_1, \ldots, u_m\}\) when \(n = 0\). [\(\blacksquare\)]
Consider the formula \( \varphi[k, e] \) where \( \varphi[k, e] \) is \( \bigvee_{i=1}^{n} e < \ell_i \lor \bigvee_{i=1}^{m} e > u_i \), for \( n > 0 \) or \( m > 0 \)

\[
S_{\text{SLRA}}(I, \Gamma, \neg \varphi[k, e]) = \begin{cases} 
\ell_j, & \text{if } n > 0 \text{ and } \max(\ell_1^1, \ldots, \ell_n^1) = \ell_j^1 \\
u_i, & \text{if } n = 0 \text{ and } \min(u_1^1, \ldots, u_m^1) = u_i^1
\end{cases}
\]

\textbf{Fig. 2.} A selection function \( S_{\text{SLRA}} \) for a simple fragment of LRA.

\textbf{Lemma 5} \( S_{\text{SLRA}} \) is monotonic for \( \varphi[k, e] \).

\textbf{Proof:} Let \( \mathcal{I} \) be a model of LRA and \( \Gamma \), where \( \Gamma \models \varphi[k, t] \), and assume by contradiction \( S_{\text{SLRA}} \) returns \( t \). Consider the case where \( n > 0 \) and \( t = \ell_i \) for some \( i \in \{1, \ldots, n\} \). Since \( \mathcal{I} \models \Gamma \cup \neg \varphi[k, e] \), it satisfies:

\[
(\ell_1 < \ell_1 \lor \ldots \lor \ell_i < \ell_i \lor \ell_i > u_1 \lor \ldots \lor \ell_i > u_m) \land 
\]

\[
e \geq \ell_i \land \ldots \land e \geq \ell_m \land e \leq u_1 \land \ldots \land e \leq u_m
\]

(6)

Assume \( \mathcal{I} \) satisfies \( \ell_i > u_k \) for some \( k \in \{1, \ldots, m\} \). We have \( \mathcal{I} \) also satisfies \( e \geq \ell_i \) and \( e \leq u_k \), and thus \( \mathcal{I} \) must satisfy \( \ell_i < \ell_k \) for some \( \ell_k \in \{1, \ldots, n\} \), and thus \( \max(\ell_1^1, \ldots, \ell_n^1) \neq \ell_k^1 \). Thus, \( S_{\text{SLRA}}(\Gamma) \neq \ell_k \). By symmetrical reasoning when \( n = 0, m > 0 \) and \( t = u_j \) for some \( j \in \{1, \ldots, m\} \), we have that \( S_{\text{SLRA}}(\Gamma) \neq u_j \). Thus, \( S_{\text{SLRA}} \) does not return \( t \), and thus is monotonic for \( \varphi[k, e] \).

\textbf{Example 1.} Consider the formula \( \forall x(x < b \lor x > a) \). The negated skolemized form \( \neg \varphi[k, e] \) in Figure 1) of this formula is equivalent to the formula \( b \leq e \land e \leq a \) where \( e \) is a fresh constant. A possible run of \( P_{\text{SLRA}} \) on this input is as follows, where \( \Gamma \) is initially \( \emptyset \) and on each iteration \( I'' = \Gamma \cup \{b \leq e \land e \leq a\} \).

<table>
<thead>
<tr>
<th>#</th>
<th>( I' )</th>
<th>( I'' )</th>
<th>( t[k] )</th>
<th>Add to ( I' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sat</td>
<td>sat</td>
<td>( \max(b^a, b^b) = b^a )</td>
<td>( b &lt; b \lor b &gt; a )</td>
</tr>
<tr>
<td>2</td>
<td>sat</td>
<td>unsat</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In step 2, note that \( I'' = \{b < b \lor b > a, b \leq e \land e \leq a\} \), which is unsat. The run establishes that \( \exists ab \forall x(x > a \lor x < b) \) is LRA-satisfiable.

\textbf{Example 2.} Consider the formula \( \forall x(x < a \land x < b) \), whose skolemized negation is equivalent to \( e \geq a \land e \geq b \). A possible run of \( P_{\text{SLRA}} \) on this input is as follows.

<table>
<thead>
<tr>
<th>#</th>
<th>( I' )</th>
<th>( I'' )</th>
<th>( t[k] )</th>
<th>Add to ( I' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sat</td>
<td>sat</td>
<td>( \max(a^a, b^b) = a^a )</td>
<td>( a &lt; a \lor a &lt; b )</td>
</tr>
<tr>
<td>2</td>
<td>sat</td>
<td>sat</td>
<td>( \max(a^a, b^b) = b^b )</td>
<td>( b &lt; a \lor b &lt; b )</td>
</tr>
<tr>
<td>3</td>
<td>unsat</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that the formula added in step 1, equivalent to \( a < b \), ensures that, in the second iteration, \( \max(a^a, b^b) \neq a^a \). The run establishes that \( \exists ab \forall x(x > a \lor x < b) \) is LRA-unsatisfiable, as expected in a linear order without endpoints.
3 Instantiation for Quantifier-Free LRA-Formulas

Consider the case where $a$ and $x$ are vectors of Real variables and $L$ is the class of formulas $\exists a \forall x \varphi[a, x]$ where $\varphi[a, x]$ is an arbitrary quantifier-free LRA-formula. We assume that equalities are eliminated from $\varphi$ by the transformation:

$$t \approx 0 \rightarrow 0 \leq t \land 0 \geq t$$

Figure 3 gives a selection function $\Delta_{\text{LRA}}$ for LRA, which takes an interpretation $I$, a set of formulas $\Gamma$, and the formula $\neg \varphi[k, e]$. It invokes the recursive procedure $S_R$ which constructs a term corresponding to each variable in $e$. Analogous to existing approaches for linear quantifier elimination [33, 40], our approach makes use of non-standard terms for symbolically representing substitutions. In particular, the terms we consider may involve a free distinguished constant $\delta$, representing an infinitesimal positive value. For each variable $e_i$ from $e$, 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 $M$ which propositionally entail $\psi$ and are satisfied by $I$, which we call a propositionally satisfying assignment for $\psi$. We partition $M$ into three sets $M_0, M_u$ and $M_c$, where $M_0$ 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 $M_0$ and $M_u$ are equivalent to sets of literals that are in solved form with respect to $e$. When $M_0$ contains at least one literal, we may return the lower bound whose value is maximal according to $I$, and similarly for $M_u$. If both $M_0$ and $M_u$ are empty, we return the term 0. When $S_{R0}$ returns the term $t_i$, we apply the substitution $\{e_i \rightarrow t_i\}$ to $\psi$ and $t$, and append $t_i$ to $t$. Terms returned by $S_{R0}$ may involve the constant $\delta$. We define a satisfiability relation between models and formulas involving $\delta$ as well as the max and min function for terms involving $\delta$ in the obvious way, such that $(t_1 + c_1 \cdot \delta)^2 > (t_2 + c_2 \cdot \delta)^2$ if either $t_1^2 > t_2^2$ or both $t_1^2 = t_2^2$ and $c_1 > c_2$.

Overall, $\Delta_{\text{LRA}}$ returns a tuple of terms $t$, after which we add the instance $\varphi[k, t]$, to $\Gamma$ in Figure 1. We assume $\downarrow$ eliminates occurrences of $\delta$ by the following transformations, which is inspired by virtual term substitution [33]:

$$\delta < t \rightarrow 0 < t \land \delta > t \rightarrow 0 \geq t \text{ where } \delta \notin \text{FV}(t).$$

Lemma 6 $\Delta_{\text{LRA}}$ is finite for $\varphi[k, e]$.

Proof: We first show that only a finite number of terms can be returned by $S_{R0}(I, (\neg \varphi[k, e]) \sigma, e_i)$ for any $I, \sigma$. Let $A$ be the set of atoms occurring in $\varphi[k, e] \sigma$. The literals in satisfying assignments of $(\neg \varphi[k, e]) \sigma$ are over these atoms. Let $\{e_i < \ell_1, \ldots, e_i < \ell_n, e_i > u_1, \ldots, e_i > u_m\}$ be the set of atoms that are in solved form with respect to $e_i$, that are equivalent to the atoms of $A$ containing $e_i$, where $e_i \notin \text{FV}(\ell_1, \ldots, \ell_n, u_1, \ldots, u_m)$. The terms returned by $S_{R0}(I, (\neg \varphi[k, e]) \sigma, e_i)$ are in $\{0, \ell_1(\pm \delta), \ldots, \ell_n(\pm \delta), u_1(\pm \delta), \ldots, u_m(\pm \delta)\}$. Since there are only a finite number of recursive calls to $S_R$ within $\Delta_{\text{LRA}}$, and each call appends only a finite number of possible terms to $t$, the set of possible return values of $\Delta_{\text{LRA}}$ is finite, and thus it is finite for $\varphi[k, e]$.  

\[\square\]
A selection function $S_{\text{LRA}}(I, \psi, \neg \varphi[k, e])$:

Return $S_R(I, \neg \varphi[k, e], e, \{\})$

$S_R(I, \psi, (e_1, \ldots, e_n), t)$:

If $i > n$, return $t$
Otherwise, let $t_i = S_R(I, \psi, e_i), \sigma = \{e_i \mapsto t_i\}$
Return $S_R(I, \psi, (e_{i+1}, \ldots, e_n), (t \sigma, t_i))$

Let $M = M_0 \cup M_{n} \cup M_e$ be such that:
- $I \vDash M$ and $M \models_p \psi$.
- $M_0 \models \{e \equiv \ell_1, \ldots, e \equiv \ell_n\}$.
- $M_n \models \{e \not< u_1, \ldots, e < u_m\}$, and
- $e \not\in \text{FV}(\ell_1, \ldots, \ell_n) \cup \text{FV}(u_1, \ldots, u_m) \cup \text{FV}(M_e)$.

Return one of

\[
\begin{align*}
\ell_i + \delta_i^n &> 0, \max\{(\ell_1 + \delta_1^n)^2, \ldots, (\ell_n + \delta_n^n)^2\} = (\ell_i + \delta_i^n)^2 \\
0 &< 0 \\
0 &< n = 0 \text{ and } m = 0
\end{align*}
\]

Lemma 7 If $I$ is a model for LRA and quantifier-free formula $\psi$, then $I$ is also a model for $\psi[e \mapsto S_{R_0}(I, \psi, e)]$.

Proof: Let $M$ be a set of literals of the form described in the definition of $S_{R_0}$ for $I, \psi$, and $e$. Consider the case where $S_{R_0}(I, \psi, e) = \ell_i + \delta_i^n$ for some $i$, where $n > 0$. We show that $I$ satisfies $M \{e \mapsto \ell_i + \delta_i^n\}$. First, since $\max\{(\ell_1 + \delta_1^n)^2, \ldots, (\ell_n + \delta_n^n)^2\} = (\ell_i + \delta_i^n)^2$, we know that $I$ satisfies $M_e \{e \mapsto \ell_i + \delta_i^n\}$. In the case that the bound on $e$ we consider is strict, that is, $e > \ell_i \in M_I$, then $\delta_i^n$ is $d_i^n$, and $\ell_i^n < u_j^n$ for all $j \in \{1, \ldots, m\}$. Thus, $I$ satisfies $(\ell_i + d_i^n < u_j^n) \{e \mapsto \ell_i + \delta_i^n\}$. In the case that the bound on $e$ we consider is non-strict, that is, if $e \equiv \ell_i \in M_I$, then $\delta_i^n$ is $0$, and $\ell_i^n \leq u_j^n$ for all $j \in \{1, \ldots, m\}$. Thus, $I$ satisfies $(\ell_i \equiv u_j) \{e \mapsto \ell_i\}$ $M_e \{e \mapsto \ell_i + \delta_i^n\}$. Finally, $I$ clearly satisfies $M_e \{e \mapsto \ell_i + \delta_i^n\} = M_e$. 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 $\psi$ does not contain $e$, and $I$ satisfies $M \{e \mapsto 0\}$. In each case, $I$ satisfies $M \{e \mapsto S_{R_0}(I, \psi, e)\}$, which entails $\psi \{e \mapsto S_{R_0}(I, \psi, e)\}$, and thus the lemma holds.

Lemma 8 $S_{\text{LRA}}$ is model-preserving for $\varphi[k, e]$.

Proof: By the definition of $S_R$ and repeated applications of Lemma 7.

Theorem 2 $P_{S_{\text{LRA}}}$ is a sound and complete procedure for determining the LRA-satisfiability of $\exists a \forall x \varphi[a, x]$. 

Fig. 3. A selection function $S_{\text{LRA}}$ for arbitrary quantifier-free LRA-formula $\varphi[a, x]$. Each $\prec$ is either $< \text{ or } \leq; \delta_i^n$ is $\delta$ if the $i^\text{th}$ lower bound for $e$ is strict, and $0 \text{ otherwise. Similarly, each } \succ$ is either $> \text{ or } \geq; \delta_i^n$ is $\delta$ if the $j^\text{th}$ upper bound for $e$ is strict, and $0 \text{ otherwise.}$
Proof: By Theorem 1 and Lemma 3 of our framework as well as LRA-specific Lemma 6 and Lemma 8.

We illustrate the procedure through examples. \(S_{R0}\) is non-deterministic; we choose instantiations only based on the lower bounds \(M_e\) found in the procedure \(S_{R0}\), though the procedure is free to base its instantiations on the upper bounds \(M_u\) as well. We underline the literal in \(M_e\) corresponding to the bound whose value is maximal in \(I\). \(I'\) is initially empty and on each iteration \(I'\) is the union of \(I\) and the skolemized negation of the input formula. Each round of \(\mathcal{S}_{\text{LRA}}\) computes a tuple \(t[k]\), which is used to instantiate our quantified formula in Figure 1. The last column shows the corresponding instance of the quantified formula after simplification, including the elimination of \(\delta\).

Example 3. To demonstrate how non-strict bounds are handled, consider the formula \(\forall x (x \leq a \lor x \leq b)\), whose skolemized negation is \(e > a \land e > b\). A possible run of \(\mathcal{P}_{\text{LRA}}\) on this input is as follows.

<table>
<thead>
<tr>
<th>#</th>
<th>(\Gamma)</th>
<th>(\Gamma')</th>
<th>(e)</th>
<th>(M_e)</th>
<th>return</th>
<th>(t[k])</th>
<th>Add to (\Gamma)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sat</td>
<td>sat</td>
<td>({e &gt; a, e &gt; b})</td>
<td>(a + \delta)</td>
<td>((a + \delta))</td>
<td>(a &lt; b)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>sat</td>
<td>sat</td>
<td>({e &gt; a, e &gt; b})</td>
<td>(b + \delta)</td>
<td>((b + \delta))</td>
<td>(b &lt; a)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>unsat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This run shows \(\exists ab \forall x (a \geq x \lor x \geq b)\) is LRA-unsatisfiable. The disjuncts of the instance \(a + \delta \leq a \land a + \delta \leq b\) added to \(\Gamma\) on the first iteration simplify to \(\perp\) and \(a < b\) respectively. We similarly obtain \(b < a\) on the second iteration.

Example 4. To demonstrate how multiple universally quantified variables are handled, consider the formula \(\forall xy (x + y < a \lor x - y < b)\) whose skolemized negation is \(e_1 + e_2 \geq a \land e_1 - e_2 \geq b\). A possible run of \(\mathcal{P}_{\text{LRA}}\) is as follows.

<table>
<thead>
<tr>
<th>#</th>
<th>(\Gamma)</th>
<th>(\Gamma')</th>
<th>(e)</th>
<th>(M_e)</th>
<th>return</th>
<th>(t[k])</th>
<th>Add to (\Gamma)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sat</td>
<td>sat</td>
<td>(e_1, e_2 \geq a - e_2, e_1 &gt; b + e_2)</td>
<td>(b + e_2)</td>
<td>(a - b)</td>
<td>((a + b, a - b))</td>
<td>(\perp)</td>
</tr>
<tr>
<td>2</td>
<td>unsat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This run shows \(\exists ab \forall xy (x + y < a \lor x - y < b)\) is LRA-unsatisfiable. The substitution for \(e_2\) is chosen based on \(M_e\) after applying the substitution \(\{e_1 \mapsto b + e_2\}\).

Example 5. To demonstrate how quantified formulas with Boolean structure are handled, consider the formula \(\forall x ((a < x \land x < b) \lor x < a + b)\) whose skolemized negation is \(a \geq e \land e \geq a + b\). A possible run of \(\mathcal{P}_{\text{LRA}}\) is as follows.

<table>
<thead>
<tr>
<th>#</th>
<th>(\Gamma)</th>
<th>(\Gamma')</th>
<th>(e)</th>
<th>(M_e)</th>
<th>return</th>
<th>(t[k])</th>
<th>Add to (\Gamma)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sat</td>
<td>sat</td>
<td>(e \geq a + b)</td>
<td>(a + b)</td>
<td>((a + b))</td>
<td>(0 &lt; b \land a &lt; 0)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>sat</td>
<td>sat</td>
<td>(e \geq b, e \geq a + b)</td>
<td>(b)</td>
<td>((b))</td>
<td>(0 &lt; a)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>unsat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This run shows $\exists ab \forall x ((a < x \land x < b) \lor x < a + b)$ is LRA-unsatisfiable. On the first iteration, we assume that the propositionally satisfying assignment for $I'$ included $a \geq e$, and hence $e \geq b$ is not included as a lower bound on that iteration. On the second iteration, the solver must satisfy both $b > 0$ and $a < 0$, which implies the model $I$ is such that $(a + b)^2 > a^2$ hence $e \geq b$ must exist in $M$, and moreover $b^2 > (a + b)^2$, hence $b$ must be the maximal lower bound for $e$.

Example 6. 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 $P_{SLRA}$ on this input is as follows.

<table>
<thead>
<tr>
<th>#</th>
<th>$\Gamma'$</th>
<th>$S_{R_0}(I, \Gamma, e)$</th>
<th>$t[k]$</th>
<th>Add to $\Gamma'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sat sat</td>
<td>$e_1 {e_1 &gt; e_2}$</td>
<td>$\emptyset$</td>
<td>$0$</td>
</tr>
<tr>
<td>2</td>
<td>unsat</td>
<td>$e_2 {\emptyset}$</td>
<td>$\delta$</td>
<td>$(0, 0)$</td>
</tr>
</tbody>
</table>

This run shows $\forall xy x > y$ is LRA-unsatisfiable. Notice that after the substitution $\{e_1 \mapsto e_2 + \delta\}$, we have that $I'$ contains neither an upper nor a lower bound for $e_2$, and hence we choose to return the value 0.

Example 7. To demonstrate a non-trivial case using the infinitesimal $\delta$, consider the formula $\forall xy (x \leq 0 \lor y - 2 \cdot x \leq 0)$ whose skolemized negation is $e_1 > 0 \land e_2 - 2 \cdot e_1 > 0$.

A possible run of $P_{SLRA}$ on this input is as follows.

<table>
<thead>
<tr>
<th>#</th>
<th>$\Gamma'$</th>
<th>$S_{R_0}(I, \Gamma, e)$</th>
<th>$t[k]$</th>
<th>Add to $\Gamma'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sat sat</td>
<td>$e_1 {e_1 &gt; 0}$</td>
<td>$\delta$</td>
<td>$(\delta, 0)$</td>
</tr>
<tr>
<td>2</td>
<td>unsat</td>
<td>$e_2 {e_2 &gt; 2 \cdot \delta}$</td>
<td>$3 \cdot \delta$</td>
<td>$(\delta, 3 \cdot \delta)$</td>
</tr>
</tbody>
</table>

This run shows $\forall xy x \leq 0 \lor y - 2 \cdot x \leq 0$ is LRA-unsatisfiable.

The procedure $P_{SLRA}$, which is an instance of the procedure in Figure 1, can be understood as lazily enumerating the disjuncts of the Loos-Weispfenning method for quantifier elimination over linear real arithmetic [33], with minor differences. In this way, our approach is similar to the projection-based procedures described in [12, 29]. These approaches compute implicants of quantified formulas, while our approach instead computes a term which is in turn used for instantiation. This choice is important for our purposes, as it enables a uniform combination of the approach with existing instantiation-based techniques for first-order logic [17, 22, 49], and allows the approach to be used as a subprocedure for synthesis as described in [47].

An alternative return value for $S_{R_0}$ is presented in Figure 4. When using this return value, the procedure $P_{SLRA}$ enumerates the disjuncts of Ferrante and Rackoff’s method for quantifier elimination over linear real arithmetic [21], with minor differences. We provide an experimental evaluation of both of these selection functions in Section 6.

---

4 For instance, that method uses a distinguished term $\infty$ representing an arbitrarily large positive value.
An Instantiation-Based Approach for Solving Quantified Linear Arithmetic

\[
\begin{array}{ll}
\frac{u_j + \delta}{2} & n > 0 \text{ and } m > 0 \\
\ell_i + 1 & n > 0 \text{ and } m = 0 \\
u_j - 1 & n = 0 \text{ and } m > 0 \\
0 & n = 0 \text{ and } m = 0
\end{array}
\]

where \( \max\{e_I^1, \ldots, e_I^n\} = e_I^j \) if \( n > 0 \)

\[
\min\{u_I^1, \ldots, u_I^m\} = u_I^j \text{ if } m > 0.
\]

Fig. 4. An alternative return value for \( S_{R0} \).

\[
\begin{array}{ll}
l_j + \delta & n > 0 \\
u_j - \delta & m > 0 \\
\infty & m = 0 \\
-\infty & n = 0
\end{array}
\]

where \( \max\{e_I^1, \ldots, e_I^n\} = e_I^j \) if \( n > 0 \)

\[
\min\{u_I^1, \ldots, u_I^m\} = u_I^j \text{ if } m > 0.
\]

Fig. 5. An alternative return value for \( S_{R0} \) that is analogous to Loos and Weispfenning’s method.

3.1 Comparison to Existing Approaches

As mentioned, at their core, most approaches for solving quantified linear arithmetic (including ours) share many similarities with one another. In particular, given an existentially quantified formula \( \exists x . \varphi \), 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 Figure 3 involves the use of a free distinguished constant \( \delta \), representing an infinitessmal positive value. Other approaches also involve use of a free distinguished constant \( \infty \), representing an arbitrarily large positive value. Like \( \delta \), this term can be eliminated, as follows:

\( \infty < t \leadsto \bot \) and \( \infty > t \leadsto \top \) where \( \infty \not\in \text{FV}(t) \).

The two most widely known algorithms for quantifier elimination for linear real arithmetic are the method based on infinitessimals in [33], and the method based on an interior point method in [21]. To put these algorithms into the context of our approach, we provide two additional alternatives for the return value of \( S_{R0} \) (Figures 5 and 6) that closely approximate the effect of these methods.

Recent approaches are inspired by of one (or both) of these methods. The approaches described in [12, 29] are closely based on the Loos-Weispfenning method, and the approach described in [18] is closely based on Ferrante-Rackoff method. The approach described in [40] examines a certified version of both approaches.

As mentioned, Figure 3 is inspired by the Loos-Weispfenning method, but does not use infinities. Similarly, the return value in Figure 4 is inspired by Ferrante-Rackoff method, but does not use infinities. One possible advantage of the approach in Figure 4 is in the context of quantifier alternation. In particular, that selection function does not use any virtual terms, and thus we may consider instances of quantified formulas having nested quantification where eliminating virtual terms is not obvious. A complete
we find the lower bound of the form $c$ propositionally satisfying assignment for $\psi$ where $\theta$ 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 $S_{R0}$ is similar to the procedure $S_{R0}$ in the previous section, where we find the lower bound of the form $c_i \cdot e \geq \ell_i$ such that the (rational) value $(\ell_i) I$ is maximal, and similarly for $M_a$. Additionally, $S_{R0}$ adds a constant $\rho$ to the maximal lower bound (resp. minimal lower bound). This constant ensures that the returned term $t_i$ and $e$ are congruent modulo $\theta \cdot c$ in $I$, a fact which in part suffices to show the overall function to be model-preserving. It then constructs a substitution with coefficients $\sigma$ of the form $\{c, e_1 \mapsto t_1\}$. A substitution of this form may be applied to integer terms of the form $c \cdot (d \cdot e_i + s)$ where $e_i \not\in FV(s)$, where $(c \cdot (d \cdot e_i + s))\sigma$ is defined as $d \cdot t_i + c \cdot s$. Additionally, we define $(s_1 \join s_2)\sigma$ as $(c \cdot s_1)\sigma \join (c \cdot s_2)\sigma$ for $\join \in \{\lt, \gt\}$, and thus we can apply $\sigma$ to arbitrary LIA-formulas. After constructing $\sigma$, the procedure $S_I$ invokes a recursive call where $\sigma$ is applied to $\psi$ and $(c \cdot t)$, $\theta$ is multiplied by $c$, the term $\theta \cdot t_i$ is appended to $t$, and $p_i$ is appended to $p$.

Overall, $S_I$ returns a vector of terms $(t \div p \theta)$, that is, integer division applied pairwise to the terms in $t$ and the constant $\theta$, where $p$ determines whether this division rounds up or down. We add the instance $\varphi[k, t \div p \theta] \downarrow$ to $I$ in Figure 1, where occurrences of integer division are eliminated by defining $\varphi[k, t \div p \theta] \downarrow$ as $\varphi[k, d] \land \theta \cdot d \equiv t \div p m \land 0 \leq m < \theta$, where $d$ and $m$ are distinct fresh constants, and $\div$ is + if $p$ is + and analogously for -. Note that our selection function chooses $p$ 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 9** $S_{LIA}$ is finite for $\varphi[k, e]$. 

Fig. 6. An alternative return value for $S_{R0}$ that is analogous to Ferrante and Rackoff’s method.

strategy for quantifier alternation using this selection function is the subject of future work.

**4 Instantiation for Quantifier-Free LIA-Formulas**

We now turn our attention to the class of arbitrary LIA-formulas $\exists a \forall x \varphi[a, x]$, $a$ and $x$ are vectors of Int variables, and where $\varphi[a, x]$ is quantifier-free. We again assume all equalities are eliminated from $\varphi$ by replacing them with a conjunction of inequalities.

Figure 7 gives a selection function $S_{LIA}$ for LIA. The procedure invokes the recursive procedure $S_I$, which takes as arguments $I$, $\neg \varphi[k, e]$, variables $e$ that we have yet to incorporate into the substitutions, an integer $\theta$, terms $t$ found as substitutions for variables from $e$ so far, and a tuple of symbols $p$ from $\{+,-\}$ which we refer to as polarities. The role of $\theta$ will be to capture divisibility relationships through the procedure, where $\theta$ is initially 1. The procedure invokes a call to $S_{R0}(I, \psi, e_i)$ which based on the propositionally satisfying assignment for $\psi$ 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 $S_{R0}$ is similar to the procedure $S_{R0}$ in the previous section, where we find the lower bound of the form $c_i \cdot e \geq \ell_i$ such that the (rational) value $(\ell_i) I$ is maximal, and similarly for $M_a$. Additionally, $S_{R0}$ adds a constant $\rho$ to the maximal lower bound (resp. minimal lower bound). This constant ensures that the returned term $t_i$ and $e$ are congruent modulo $\theta \cdot c$ in $I$, a fact which in part suffices to show the overall function to be model-preserving. It then constructs a substitution with coefficients $\sigma$ of the form $\{c, e_1 \mapsto t_1\}$. A substitution of this form may be applied to integer terms of the form $c \cdot (d \cdot e_i + s)$ where $e_i \not\in FV(s)$, where $(c \cdot (d \cdot e_i + s))\sigma$ is defined as $d \cdot t_i + c \cdot s$. Additionally, we define $(s_1 \join s_2)\sigma$ as $(c \cdot s_1)\sigma \join (c \cdot s_2)\sigma$ for $\join \in \{\lt, \gt\}$, and thus we can apply $\sigma$ to arbitrary LIA-formulas. After constructing $\sigma$, the procedure $S_I$ invokes a recursive call where $\sigma$ is applied to $\psi$ and $(c \cdot t)$, $\theta$ is multiplied by $c$, the term $\theta \cdot t_i$ is appended to $t$, and $p_i$ is appended to $p$.

Overall, $S_I$ returns a vector of terms $(t \div p \theta)$, that is, integer division applied pairwise to the terms in $t$ and the constant $\theta$, where $p$ determines whether this division rounds up or down. We add the instance $\varphi[k, t \div p \theta] \downarrow$ to $I$ in Figure 1, where occurrences of integer division are eliminated by defining $\varphi[k, t \div p \theta] \downarrow$ as $\varphi[k, d] \land \theta \cdot d \equiv t \div p m \land 0 \leq m < \theta$, where $d$ and $m$ are distinct fresh constants, and $\div$ is + if $p$ is + and analogously for -. Note that our selection function chooses $p$ 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.
\[ S_{\text{LIA}}(\mathcal{I}, \psi, \neg \varphi[k, e]) : \]

Return \( S_I(\mathcal{I}, \neg \varphi[k, e], e, 1, \{\}) \)

\( S_I(\mathcal{I}, \psi, (e_1, \ldots, e_n), \theta, t, p) : \)

If \( i > n \), return \( t \mod \theta \)

Otherwise, let \( (c, t_i, p_i) = S_{I0}(\mathcal{I}, \psi, \theta, \sigma = \{c \cdot e_i \mapsto t_i\}) \)

Return \( S_I(\mathcal{I}, \psi \sigma, (e_{i+1}, \ldots, e_n), \theta \cdot c, ((c \cdot t) \sigma, \theta \cdot t_i, (p, p_i)) \)

\( S_{I0}(\mathcal{I}, \psi, \theta) : \)

Let \( M = M_\ell \cup M_u \cup M_c \) be such that:

- \( \mathcal{I} \models M \) and \( M \models \psi \),
- \( M_\ell \models \{c_1 \cdot e \geq \ell_1, \ldots, e \geq \ell_n\}, e_1 > 0, \ldots, e_n > 0, \)
- \( M_u \models \{d_1 \cdot e \leq u_1, \ldots, d_m \cdot e \leq u_m\}, d_1 > 0, \ldots, d_m > 0, \) and
- \( e \notin \text{FV}(\ell_1, \ldots, \ell_n) \cup \text{FV}(u_1, \ldots, u_m) \cup \text{FV}(M_c) \).

Return one of

\[
\begin{align*}
(c_i, \ell_i + \rho, +) & \quad n > 0, \max\left(\frac{c_i}{\ell_i}\right)^2, \rho = (c_i \cdot e - \ell_i)^2 \mod (\theta \cdot c_i) \\
(d_j, u_j - \rho, -) & \quad m > 0, \min\left(\frac{d_j}{u_j}\right)^2, \rho = (u_j - d_j \cdot e)^2 \mod (\theta \cdot d_j) \\
(1, \rho, +) & \quad n = 0, m = 0, \rho = e^2 \mod \theta
\end{align*}
\]

Fig. 7. A selection function \( S_{\text{LIA}} \) for arbitrary quantifier-free LIA-formula \( \varphi[a, x] \).

**Proof:** First, we show that only a finite number of tuples are returned by \( S_{I0}(\mathcal{I}, \neg \varphi[k, e]) \) for any \( \mathcal{I}, \sigma, e_i, \theta \) with \( \theta > c \cdot t \).

Let \( A = \{c \cdot e_i < \ell_1, \ldots, e_i < \ell_n, d_1 \cdot e_i \geq u_1, \ldots, d_m \cdot e_i > u_m\} \) be the set of quantifiers that are in solved form with respect to \( e_i \) that are equivalent to the atoms of \( A \) containing \( e_i \), where \( e_i \not\in \text{FV}(\ell_1, \ldots, \ell_n, u_1, \ldots, u_m) \) and \( e_1 > 0, \ldots, e_n > 0, d_1 > 0, \ldots, d_m > 0 \).

The tuples returned by \( S_{I0}(\mathcal{I}, \neg \varphi[k, e]) \) are in the finite set:

\[
\begin{align*}
\{c_i, \ell_i + \rho, +\} & \quad 1 \leq i \leq n \\
\{d_j, u_j + 1 + \rho, -\} & \quad 1 \leq j \leq m \\
\{d_j, u_j - \rho, -\} & \quad 1 \leq j \leq m \\
\{c_i, \ell_i - 1 - \rho, +\} & \quad 1 \leq i \leq n \\
\{(1, \rho, +)\}
\end{align*}
\]

and where \( 0 \leq \rho < (\theta \cdot c) \).

Since \( c \) and \( \theta \) are 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 \( S_{\text{LIA}} \), and each call modifies \( t \) based a finite number of possible tuples coming from the set above, the set of possible return values of \( S_{\text{LIA}} \) is finite, and thus it is finite for \( \varphi[k, e] \).

**Lemma 10** If \( \mathcal{I} \) is a model for LIA and a quantifier-free formula \( \psi, \theta \geq 1, \) and \( S_{I0}(\mathcal{I}, \psi, e, \psi \sigma, (c, t, p)) \), then:

1. \( (c \cdot e)^2 \equiv_{\theta \cdot c} t^2 \), and
2. \( \mathcal{I} \models \psi[c \cdot e \mapsto t] \).
In the case that by our selection of the form described in the body of $S_{M\sigma}$ Reynolds, King, Kuncak is equivalent to is equivalent to

\[(c_i \cdot \ell_i + \rho, +),\]

and

\[(\ell_i + \rho)^T \equiv_{\theta, c_i} (\ell_i + (c_i \cdot e - \ell_i)^T) \text{ mod } (\theta \cdot c_i)^T \equiv_{\theta, c_i} (c_i \cdot e)^T.\]

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

\[(u_j - \rho)^T \equiv_{\theta, d_j} (u_j - (u_j - d_j \cdot e))^T \text{ mod } (\theta \cdot d_j)^T \equiv_{\theta, d_j} (d_j \cdot e)^T.\]

In the case that $n = 0, m = 0$, and $S_{I0}(\mathcal{I}, \psi, e, \theta) = (1, \rho, +)$, we have that $\rho^T \equiv_{\theta, 1} (e^T \text{ mod } \theta)^T \equiv_{\theta, 1} (1 \cdot e)^T$.

To show part 2, we first focus on the case where $n > 0$ and $S_{I0}(\mathcal{I}, \psi, e, \theta) = (c_i, \ell_i + \rho, +)$. We have that $\rho = (c_i \cdot e - \ell_i)^T \text{ mod } (\theta \cdot c_i)$. Let $M$ be a set of literals of the form described in the body of $S_{I0}(\mathcal{I}, \psi, e)$. We show that $\mathcal{I}$ satisfies each literal in $M\sigma$, where $\sigma = \{c_i \cdot e \mapsto \ell_i + \rho\}$. First, consider an atom in $M\sigma$ that is equivalent to $(c_j \cdot e \mapsto \ell_j)$ for some $j \in \{1, \ldots, n\}$. This is equivalent to $(c_j \cdot e \mapsto \ell_j)\sigma$, which is equivalent to $c_j \cdot c_i \cdot (\ell_i + \rho) \equiv (c_j \cdot c_i) \cdot \ell_j$, which is satisfied by $\mathcal{I}$ since $(c_j \cdot c_i) \equiv (c_j \cdot c_i)^T$ by our selection of $(c_i, \ell_i + \rho)$ and since $\rho \equiv 0$. Second, consider the atom in $M\sigma$ that is equivalent to $(d_j \cdot e \mapsto u_j)\sigma$ for some $j \in \{1, \ldots, m\}$. Let $\rho' = (c_i \cdot e - \ell_i)^T$, which is greater than 0 since $\mathcal{I}$ satisfies $(c_i \cdot e \mapsto \ell_i)$. Since $(c_i \cdot e)^T = (\ell_i + \rho')^T$, we have that $\mathcal{I}$ satisfies $(d_j \cdot e \mapsto u_j)\sigma$, which is equivalent to $(d_j \cdot (\ell_i + \rho') \equiv (c_i \cdot u_j)$. Since $\rho = \rho' \equiv (c_i \cdot e \mapsto \ell_i)^T$, we have that $\mathcal{I}$ also satisfies $(d_j \cdot (\ell_i + \rho) \equiv (c_i \cdot u_j)$, which is $(d_j \cdot e \mapsto u_j)\sigma$. Finally, $\mathcal{I}$ satisfies $M\sigma$ as $M\sigma = M_\sigma$ and $\mathcal{I} = M_\sigma$. Thus, $\mathcal{I}$ satisfies $M\sigma$, which entails $\psi\sigma$. The case for when $m > 0$ and $S_{I0}(\mathcal{I}, \psi, e, \theta) = (d_j, u_j - \rho, -)$ is symmetric. When $n = 0, m = 0$, and $S_{I0}(\mathcal{I}, \psi, e) = (1, \rho, +)$, the assignment $M$ does not contain $e$, and thus $\mathcal{I}$ satisfies $M\{c \mapsto e \mapsto \rho\} = M$ and $\psi\{c \mapsto e \mapsto \rho\}$.

Lemma 11 Each recursive call to $S_j(\mathcal{I}, \psi, (e_1, \ldots, e_n), \theta, (t_1, \ldots, t_{i-1}), p)$ within $S_{I\text{LA}}(\mathcal{I}, \Gamma, (e_1, \ldots, e_n))$ is such that:

1. $\theta | t_j^T$ for each $1 \leq j < i$, and
2. $\mathcal{I} = \psi$ and $\psi$ is equivalent to $\neg \varphi[k, e]\{\theta \cdot e_i \mapsto t_1\} \cdots \{\theta \cdot e_{i-1} \mapsto t_{i-1}\}$.

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

To show part 1, we have from Lemma 10 part 1 that:

\[(c \cdot e_i)^T \equiv_{\theta, e} (t_i + \rho)^T\] (7)

Consider a $t_j$ where $1 \leq j < i$, where by our assumption is such that $\theta | t_j^T$, and thus $\theta \cdot c \cdot (c \cdot t_j)^T$. By (7), we have that $\theta \cdot c \cdot ((c \cdot t_j)\sigma)^T$. Also by (7), we have that $c \cdot (t_j + \rho)^T$, and thus $\theta \cdot c \cdot ((\theta \cdot \theta \cdot d_j + \rho)^T$.

To show part 2, by our assumption, $\mathcal{I} = \psi$ and thus by Lemma 10 part 2 we have that $\mathcal{I} = \psi\sigma$. By our assumption, $\psi$ is equivalent to $\neg \varphi[k, e]\{\theta \cdot e_i \mapsto t_1\} \cdots \{\theta \cdot e_{i-1} \mapsto t_{i-1}\}$. Thus, $\psi\sigma$ is equivalent to $\neg \varphi[k, e]\{\theta \cdot c \cdot e_i \mapsto (c \cdot t_j)\sigma \cdots \{\theta \cdot c \cdot e_{i-1} \mapsto (c \cdot t_{i-1})\sigma \cdot \{(\theta \cdot c) \cdot e_i \mapsto \theta \cdot (t_i + \rho)\}$. Thus, the lemma holds.
**Lemma 12** $S_{\text{LIA}}$ is model-preserving for $\varphi[k,e]$.

**Proof:** Assume that $S_{\text{LIA}}(\mathcal{I},\Gamma, \lnot \varphi[k,e]) = t$, where $e = (e_1, \ldots, e_n)$, and $t = (t_1, \ldots, t_n)$. By Lemma 11 and the definition of $S_{\text{LIA}}$, there is a $\theta$ such that for each $i = 1, \ldots, n$, term $t_i$ is of the form $s_i \div^\theta \theta$ where $\theta | s_i^\mathcal{I}$, and $\mathcal{I} \models (\lnot \varphi[k,e]) \{ \theta \cdot e_1 \mapsto s_1 \} \cdots \{ \theta \cdot e_n \mapsto s_n \}$. Thus, $\mathcal{I}$ satisfies $(\lnot \varphi[k,e]) \{ e \mapsto t \} = \lnot \varphi[k,t]$, and thus $S_{\text{LIA}}$ is model-preserving for $\varphi[k,e]$.

**Theorem 3** $P_{\text{SILIA}}$ is a sound and complete procedure for determining the LIA-satisfiability of $\exists a \forall x \varphi[a,x]$.

**Proof:** By Theorem 1, Lemma 3, Lemma 9 and Lemma 12.

**Example 8.** To demonstrate a case involving a substitution with coefficients, consider the formula $\forall x y (2 \cdot x < a \lor x + 3 \cdot y < b)$ whose negation is $2 \cdot e_1 \geq a \land e_1 + 3 \cdot e_2 \geq b$.

A possible run of $P_{\text{SILIA}}$ on this input is as follows.

<table>
<thead>
<tr>
<th>#</th>
<th>$\Gamma'$</th>
<th>$\Gamma''$</th>
<th>$e$</th>
<th>$\theta$</th>
<th>$S_{\text{LIA}}(\mathcal{I},\Gamma,e,\theta)$</th>
<th>$M_i$</th>
<th>$t[k]$</th>
<th>Add to $\Gamma'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sat</td>
<td>sat</td>
<td>$e_1$</td>
<td>1</td>
<td>${2 \cdot e_1 \geq a, \ldots}$</td>
<td>(2, a, +)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>unsat</td>
<td></td>
<td>$e_2$</td>
<td>2</td>
<td>${6 \cdot e_2 \geq 2 \cdot b - a}$</td>
<td>(6, 2 \cdot b - a, +)</td>
<td>(6 \cdot a, 4 \cdot b - 2 \cdot a)</td>
<td>div 12 \psi_1</td>
</tr>
</tbody>
</table>

Thus, $\exists a b \forall x y (2 \cdot x < a \lor x + 3 \cdot y < b)$ is LIA-unsatisfiable. We assume $\rho = 0$ for all calls to $S_{10}$ 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 $\Gamma'$ the instance $\psi_1$, which is equivalent to $2 \cdot ((6 \cdot a) \text{ div}^+12) < a \lor (6 \cdot a) \text{ div}^+12 + 3 \cdot ((4 \cdot b - 2 \cdot a) \text{ div}^+12) < b$. Applying normalization 1, to this formula results in a one that is LIA-unsatisfiable.

**Example 9.** To demonstrate a case involving a non-zero value of $\rho$, consider the formula $\forall x y (3 \cdot x + y \neq a \lor 0 > y \lor y > 2)$ whose negation is $3 \cdot e_1 + e_2 \approx a \land 0 \leq e_2 \land e_2 \leq 2$, where $\approx$ denotes the conjunction of non-strict upper and lower bounds. A possible run of $P_{\text{SILIA}}$ on this input is as follows.

<table>
<thead>
<tr>
<th>#</th>
<th>$\Gamma'$</th>
<th>$\Gamma''$</th>
<th>$e$</th>
<th>$\theta$</th>
<th>$S_{\text{LIA}}(\mathcal{I},\Gamma,e,\theta)$</th>
<th>$M_i$</th>
<th>$t[k]$</th>
<th>Add to $\Gamma'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sat</td>
<td>sat</td>
<td>$e_1$</td>
<td>1</td>
<td>${3 \cdot e_1 \geq a - e_2}$</td>
<td>(3, a - e_2, +)</td>
<td>(a, 0)</td>
<td>div^+ 3 \psi_1</td>
</tr>
<tr>
<td>2</td>
<td>sat</td>
<td>sat</td>
<td>$e_2$</td>
<td>3</td>
<td>${e_2 \geq 0}$</td>
<td>(1, 0, +)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>sat</td>
<td>sat</td>
<td>$e_1$</td>
<td>1</td>
<td>${3 \cdot e_1 \geq a - e_2}$</td>
<td>(3, a - e_2, +)</td>
<td>(a - 1, 1)</td>
<td>div^+ 3 \psi_2</td>
</tr>
<tr>
<td></td>
<td>sat</td>
<td>sat</td>
<td>$e_2$</td>
<td>3</td>
<td>${e_2 \geq 0}$</td>
<td>(1, 1, +)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>unsat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This run shows $\exists a \forall x y (2 \cdot x < a \lor x + 3 \cdot y < b)$ is LIA-unsatisfiable. On the first iteration, we assume that $\Gamma''$ 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 $S_{10}$ we have $\rho = 0$. The instance $\psi_1$ added to $\Gamma$ on this iteration is equivalent to $3 \cdot (a \text{ div}^+ 3) \neq a$ and implies that $a^2 \neq 3$ 0 in subsequent models $\mathcal{I}$. Thus, models $\mathcal{I}$ satisfying $3 \cdot e_1 + e_2 \approx a$
are such that $e_2^I \neq 3^I 0$. On the next iteration, $I'$ is satisfied by a model, call it $I_2$, where the maximal lower bound for $e_2$ is 0. By the above reasoning and since $I_2$ satisfies $3 \cdot e_1 + e_2 \approx a$, it must be that $\rho = ((e_2 - 0)^I_2 \mod 3) \neq 0$. Assume $(e_2 - 0)^I_2 \equiv 3^I$. The instance $\psi_2$ is equivalent to $3 \cdot ((a - 1) \div 3^I + 1 \neq a$, which implies that $a^I \neq 3^I 1$ in subsequent models $I$, and hence $e_2^I \neq 3^I 1$. The instance $\psi_3$ is equivalent to $3 \cdot ((a - 2) \div 3^I + 2 \neq a$ and implies that $a^I \neq 3^I 2$, which together with the two previous instances are $T$-unsatisfiable. 

The procedure $PSLIA$ can be understood to lazily enumerating disjuncts of Cooper’s algorithm for quantifier elimination over linear integer arithmetic [15], with minor differences. The algorithm is essentially enumerating a single path of [15] by using the model to select a satisfied case split for each variable over an entire block of quantifiers. 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 $S_{10}$ is proportional to the size of $\theta$. While not shown here, our implementation takes steps to reduce the size of $\theta$ by factoring out common divisors in $\theta$ and the coefficients returned by $S_{10}$.

### 4.1 Comparison to Existing Approaches

The approach taken in $PSLIA$ is similar to the one taken in Section 2.5 in [12]. The most substantial difference between the two algorithms is that $PSLIA$ implements a variant of Cooper’s algorithm while resolve in [12] uses the model to guide an execution of the Omega test [44]. The most similar aspects of the approaches are the computation of a feasible $\rho$ 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 $\rho$ 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 Integration in an SMT Solver

This section gives an overview of how the instantiation-based procedure for quantified formulas as described in Section 2 can be integrated into a solving architecture used by SMT solvers, and used in part for determining the satisfiability of inputs $\Gamma_0$ having arbitrary Boolean structure that contain any number of quantified $T$-formulas.

Figure 8 defines a procedure SMTQI which takes as input a theory $T$ and a set of ground $T$-formulas $\Gamma$ in purified form with respect to quantified formulas, that is, a formula obtained from $\Gamma_0$ by replacing all quantified formulas $\forall x. \varphi_i[x]$ in $\Gamma_0$ by (uniquely associated) boolean variables $A_i$. We call such a variable $A_i$ the positive guard of $\forall x. \varphi_i[x]$ and write $A_i \leftrightarrow \forall x. \varphi_i[x]$ to denote $A_i$ is the positive guard of $\forall x. \varphi_i[x]$. In each call to SMTQI, if $\Gamma$ is $T$-unsatisfiable, then the procedure returns

---

3In the parlance of [15], $PSLIA$ selects a feasible $j$ value using the calculation of $\rho$ and avoids introducing the $P_{2,\infty}$ cases by introducing the no bounds case ($n = 0, m = 0$) and always favoring bounds when one exists.
If \( \Gamma \) is \( T \)-unsatisfiable, then return “unsat”.

Otherwise, let \( M \) be such that \( \mathcal{I} \models M \) and \( M \models \Gamma \) for some model \( \mathcal{I} \) of \( T \) and \( \Gamma \).

\[
\Gamma' := \Gamma.
\]

For each \((\neg \phi)|_{\mathcal{E}_i}\) we add the formulas \((A_i \lor B_i) \cup (\mathcal{B} \implies \neg \phi_i[\mathcal{E}_i])\). If \( A_i \in M \) and \( B_i \in M \),

\[
\Gamma := \Gamma' \cup (A_i \implies \phi_i[S_i(\mathcal{I}, \Gamma, \neg \phi_i[\mathcal{E}_i]))] \]

If \( \Gamma' = \Gamma \), then return “sat”. Otherwise, return \( \text{SMTQI}(T, \Gamma') \).

Fig. 8. Procedure \( \text{SMTQI} \) for SMT solving with quantifier instantiation, which determines the \( T \)-satisfiability of a ground set of \( T \)-formulas \( \Gamma \) in purified form with respect to quantified formulas.

“unsat”. Otherwise, we find a model \( \mathcal{I} \) of \( T \) and \( \Gamma \) and a corresponding propositionally satisfying assignment \( M \). We then build a new set of formulas \( \Gamma' \), initially containing \( \Gamma \), as follows. For each quantified formula \( \forall x. \phi_i[x] \) whose positive guard \( A_i \) is in \( M \), we add the formulas \((A_i \lor B_i) \cup (\mathcal{B} \implies \neg \phi_i[\mathcal{E}_i])\) to \( \Gamma' \) if we have not done so already, where \( B_i \) is a fresh Boolean variable, which we call the negative guard of \( \forall x. \phi_i[x] \). If both the positive and negative guards of \( \forall x. \phi_i[x] \) are asserted positively in \( M \) (we will say such a quantified formula is active in \( M \)), we consider an instance of this quantified formula to \( \Gamma' \) based on its associated selection function \( S_i \). If \( \phi_i \) has nested quantification, this instance will contain quantified formulas, which we purify in the same manner described above. If no new formulas are added to \( \Gamma' \) in this process, then the procedure returns “sat”. Otherwise, we call \( \text{SMTQI} \) on \( \Gamma' \).

At a high level, the procedure in Figure 8 adds guarded instances of quantified formulas to an evolving set of formulas \( \Gamma \) until \( \Gamma \) is \( T \)-unsatisfiable, or a fixed point is reached. In this respect, the algorithm is similar to existing instantiation-based approaches used by SMT solvers for quantified formulas [16, 23]. However, the procedure differs from these approaches in the following ways. Firstly, when a universally quantified formula is asserted negatively in \( M \), typical approaches add the clause \((\neg A_i \implies \neg \phi_i[\mathcal{E}_i])\) to \( \Gamma \). Here, we choose to add the clauses \((A_i \lor B_i) \cup (\mathcal{B} \implies \neg \phi_i[\mathcal{E}_i])\) instead. This allows us to consider both the positive and negative versions of quantified formulas simultaneously. As such, the algorithm only adds instances of quantified formulas where both the positive and negative guards are asserted positively in \( M \). To ensure the model soundness of the approach (that is, the algorithm answers “sat” only if the input is indeed \( T \)-satisfiable), we require the following property of the set of literals \( M \) in the body of \( \text{SMTQI} \):

If no quantified formula is active in \( M \), then \( \Gamma \cup \{B_i\} \) is \( T \)-unsat for \( i = 1, \ldots, n \), where \( \{A_1, \ldots, A_n\} \) is the set of the positive guards that are asserted positively in \( M \).

In other words, the set of literals \( M \) are chosen such that, if possible, at least one of \( B_1, \ldots, B_n \) is true in \( M \). In practice, this requirement can be met in a DPLL(T)-based
SMT solver by instructing its underlying SAT solver, when it chooses a decision literal to add to \( M \), to choose one of the unassigned negative guards of quantified formulas in \( \Gamma \), if one exists, and assert it positively.

We refer to the treatment of quantified formulas in Figure 8 as \textit{counterexample-guided quantifier instantiation} [47]. A closely related approach is that of model-based quantifier instantiation [24], which like the approach described here, adds instances of quantified formulas based on models for their negations. This approach differs in its scope, in that it primarily targets quantified formulas having uninterpreted functions, whereas the approach described in Figure 8 targets quantified formulas having no uninterpreted functions. It 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 the approach described in Figure 8 uses a single instance of the SMT solver for doing these tasks simultaneously in its main solving loop.

5.1 Arbitrary Quantifier Alternation

The algorithm in Figure 8 can be used as a basis for handling quantified formulas with arbitrary quantifier alternations. Assume we rewrite formulas added to \( \Gamma' \) in the body \( \text{SMTQI} \) so they are in purified form with respect to quantified formulas, that is, we replace each quantified formula \( \forall x.\varphi_j[x] \) occurring in formulas \( B_i \Rightarrow \neg \varphi_i[e_i] \) and \( A_i \Rightarrow \varphi_i[S_i(I, \Gamma, e_i)] \) with its corresponding positive guard \( A_j \). We give an intuition of how the procedure \( \text{SMTQI} \) handles such formulas in the following. A more comprehensive description is the subject of future work.

Consider the LIA-formula \( \forall x.\neg (\forall y.x > y) \), call it \( \varphi_1 \), whose positive guard is \( A_1 \). Given the input \( \Gamma = \{ A_1 \} \), the procedure \( \text{SMTQI} \) adds the formulas \( A_1 \lor B_1 \) and \( B_1 \Rightarrow A_2 \) to \( \Gamma \), where \( A_2 \) is the positive guard for \( \forall y.e_1 > y \) (call this formula \( \varphi_2 \)) where \( e_1 \) is a fresh constant. On the second call to \( \text{SMTQI} \), the satisfying assignment includes \( A_2 \) and the procedure similarly adds the formulas \( A_2 \lor B_2 \) and \( B_2 \Rightarrow e_1 \leq e_2 \) to \( \Gamma \) where \( e_2 \) is a fresh constant. On the third call to \( \text{SMTQI} \), the satisfying assignment includes \( A_2 \ltimes B_3 \) and \( B_3 \Rightarrow e_1 \leq e_2, \ldots \) is \( T \)-satisfiable, and we may choose a satisfying assignment \( M = \{ A_1, B_1, A_2, B_2, e_1 \leq e_2, \ldots \} \). Both \( \varphi_1 \) and \( \varphi_2 \) are active in \( M \).

The literal \( e_1 \leq e_2 \) is over the atoms of \( \varphi_2[e_2/y] \), and we may add the formula \( A_2 \Rightarrow e_1 > e_1 \) to \( \Gamma \) on this iteration, assuming our selection function for \( \varphi_2 \) chose to return the maximal lower bound \( e_1 \) for \( e_2 \). On the fourth call to \( \text{SMTQI} \), we have that \( \Gamma = \{ A_1, B_1 \Rightarrow A_2, A_2 \Rightarrow e_1 > e_1, \ldots \} \), and hence \( B_1 \) or \( A_2 \) cannot be asserted positively in a satisfying assignment \( M \) for this set. Hence, neither \( \varphi_1 \) nor \( \varphi_2 \) is active in \( M \) and the procedure \( \text{SMTQI} \) adds no instances to \( \Gamma \), indicating that our input is satisfiable.

6 Experimental Evaluation

We have implemented the procedure in the SMT solver \textsc{CVC}4 [4] (version 1.5 prerelease). This section presents an evaluation of this implementation compared against other SMT solvers and first-order theorem provers.
An Instantiation-Based Approach for Solving Quantified Linear Arithmetic

### Table: Results for LRA Benchmarks

<table>
<thead>
<tr>
<th>Solver</th>
<th>Total (621)</th>
<th># time</th>
<th># time</th>
<th># time</th>
<th># time</th>
<th># time</th>
<th># time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVC4(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVC4</td>
<td></td>
<td>25</td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z3</td>
<td></td>
<td>25</td>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VampireZ3</td>
<td></td>
<td>25</td>
<td>2.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beagle</td>
<td></td>
<td>25</td>
<td>29.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vampire</td>
<td></td>
<td>25</td>
<td>1.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yices</td>
<td></td>
<td>25</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZenonArith</td>
<td></td>
<td>25</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Princess</td>
<td></td>
<td>25</td>
<td>0.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table: Results for LIA Benchmarks

<table>
<thead>
<tr>
<th>Solver</th>
<th>Total (461)</th>
<th># time</th>
<th># time</th>
<th># time</th>
<th># time</th>
<th># time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVC4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beagle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Princess</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VampireZ3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vampire</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZenonArith</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Pure Quantified Linear Arithmetic

We considered all quantified benchmarks over 6 classes in the LRA and LIA logics of the SMT library [5]. The class *keymaera* are verification conditions coming from the Keymaera verification tool [42]. *scholl* were used for simplification of non-convex polyhedra in [51], *psyco* were used for weakest precondition synthesis for compiler optimizations in [34], *uauto* correspond to verification conditions in [26], and the *tptp* classes correspond to simple arithmetic conjectures coming from the TPTP library [55]. 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 [6]. 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.  

The results for the linear real and integer benchmarks are in Figures 9 and 10 respectively. Of the 7 benchmark classes, only one (the *scholl* class from LRA) had quantified formulas with nested quantification. The algorithm in Section 5 naturally extends

---

6 Details can be found at [http://cs.uiowa.edu/~ajreynol/InstLA](http://cs.uiowa.edu/~ajreynol/InstLA).
to such formulas; a formal treatment of nested quantification is the subject of future work.

For LRA, we considered both the selection function from Figure 3, and its alternative from Figure 4, where the latter we refer to as CVC4 (a). For both LRA and LIA, the best configuration of CVC4 solves the most benchmarks overall (599 and 461 respectively), and did not give a conflicting response with any of the other solvers. Moreover, we note that CVC4 solves every benchmark that does not involve nested quantification, giving confirming evidence that our approach and implementation for solving linear arithmetic with one quantifier alternation is indeed sound and complete. Although we do not claim completeness for formulas with nested quantification, CVC4 solves more benchmarks (352) from the scholl class than any other solver.

The SMT solver Z3 (version 4.3.2), which uses the approach described in [11], solves the next most benchmarks overall, solving 573 and 455 total for the LRA and LIA sets respectively. A technique [18] in the SMT solver Yices (version 2.4.1) is able to solve all benchmarks from the keymaera and tptp classes of LRA, both does not handle quantified formulas in LIA or with nested quantification. We also considered the entrants of the first-order typed theorems division (TFA) of CASC 25, the most recent competition for automated theorem provers [56]. 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, VampireZ3, which uses a combination of a first-order theorem prover and an SMT solver [46], solves only 302 benchmarks, compared to 599 solved by CVC4 (a). The best LIA automated theorem prover, Beagle [7], solves 284 benchmarks, also notably less than the 461 solved by CVC4.

When comparing the best configuration of CVC4 to a combination of all other solvers, CVC4 solved 31 benchmarks that no other system solved, while in only 10 cases did another system solve a benchmark that CVC4 could not solve. In addition to solving the most benchmarks, CVC4 generally has small runtimes for the benchmarks it solves. When compared to the second best solver Z3, which takes 1129.8 seconds to solve 1028 benchmarks over all classes, CVC4 while solving 1060 benchmarks overall, solves its first 1028 benchmarks in a total of 159.0 seconds. Among the benchmarks solved by CVC4 and other systems, in only 12 cases did any system solve a benchmark at least 5 seconds faster than CVC4, while in 13 cases CVC4 solved a benchmark at least 5 seconds faster than all other systems.

Combining Linear Arithmetic with Uninterpreted Functions A prototype of the instantiation-based procedure from this paper was used by CVC4 in both the CASC J7 and CASC 25 competitions [56], which evaluated automated theorem provers on TPTP benchmarks involving combinations of arithmetic and free function symbols. CVC4 won the theorems (TFA) division of CASC J7 and finished 2nd in CASC 25, behind VampireZ3. Additionally, CVC4 won the non-theorems (TFN) division of CASC 25. While treatment of uninterpreted functions is beyond the scope of this work, this shows the

---

7 We omit SPASS+T, which did not handle some classes of benchmarks due to restrictions on its input format, was comparable to the other automated theorem provers for the others. We show results for an updated version of Beagle (version 0.9.30).
potential of the technique for use in general first-order automated theorem proving in the presence of background theories.

7 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.

For future work, we would like to adapt the approach for quantified linear arithmetic with arbitrary quantifier alternations, and develop heuristics for avoiding worst case performance for quantified integer arithmetic involving large coefficients. We plan to develop selection functions for other theories, in particular, algebraic datatypes and fixed-width bitvectors, as well as for combinations of theories that admit quantifier elimination. 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 Peter Baumgartner for his help with converting the benchmarks used in the evaluation to the TPTP format.

References


