Linear Arithmetic Satisfiability Via Strategy Improvement

Azadeh Farzan
University of Toronto
azadeh@cs.toronto.edu

Zachary Kincaid
Princeton University
zkincaid@cs.princeton.edu

Abstract

Satisfiability-checking of formulas in the theory of linear rational arithmetic (LRA) has broad applications including program verification and synthesis. Satisfiability Modulo Theories (SMT) solvers are effective at checking satisfiability of the ground fragment of LRA, but applying them to quantified formulas requires a costly quantifier elimination step. This article presents a novel decision procedure for LRA that leverages SMT solvers for the ground fragment of LRA, but avoids explicit quantifier elimination. The intuition behind the algorithm stems from an interpretation of a quantified formula as a game between two players, whose goals are to prove that the formula is either satisfiable or not. The algorithm synthesizes a winning strategy for one of the players by iteratively improving candidate strategies for both. Experimental results demonstrate that the proposed procedure is competitive with existing solvers.

1 Introduction

Satisfiability modulo theories (SMT) solvers have proven to be extremely effective tools for solving a variety of problems. The traditional strength of SMT solvers has been in testing satisfiability of ground (quantifier-free) formulas, but many applications require quantifiers. For example, checking verification conditions for deductive verification [Ge *et al.*, 2007], program synthesis [Solar-Lezama *et al.*, 2006; Solar-Lezama, 2008; Reynolds *et al.*, 2015], and model checking of array programs [Ghilardi and Ranise, 2010] all make use of quantifiers.

Integrating support for quantifiers into SMT solvers has been a long standing challenge. For theories that admit *quantifier elimination*, such as **linear rational arithmetic (LRA)**, one option is to eliminate quantifiers and then apply an SMT solver to the resulting ground formula. However, for applications that require only a yes or no answer to the satisfiability problem, quantifier elimination is a computationally expensive and unnecessary step. Heuristic *quantifier instantiation* is a practical alternative to quantifier elimination [De Moura and Bjørner, 2007; Ge *et al.*, 2007], but it is incomplete and may return *unknown* on difficult problem instances. First-order theorem provers (such as Vampire [Kovács and Voronkov, 2013] and E [Schulz, 2013]) are well-tuned for solving quantified formulas, but have limited support for reasoning modulo theories.

This article presents a novel procedure for checking satisfiability of (quantified) LRA formulas. As with some approaches to Quantified Boolean Formulas [Zhang, 2006; Janota *et al.*, 2012], the procedure takes intuition from game-theoretical semantics of quantifiers [Hintikka, 1982]. We interpret a quantified formula as a game played by two players, SAT and UNSAT, whose goals are to prove that the formula is satisfiable and unsatisfiable, respectively. The players take turns instantiating quantifiers in the formula, with the existential quantifiers corresponding to moves of the SAT player and universal quantifiers corresponding to moves of the UNSAT player. SAT wins the game if the choices made by the players results in a model of the remainder of the formula (after all quantifiers have been instantiated); otherwise, UNSAT wins. A quantified formula is satisfiable if and only if there is a *winning strategy* for the SAT player; that is, if SAT has a way to win the game no matter how UNSAT plays.

The decision procedure proposed in this paper is based on synthesizing a winning strategy for one of the two players. The algorithm operates by iteratively improving the strategies for both players. At each step of the algorithm, one of the players proposes a candidate strategy. If the candidate is a winning strategy, then the status of the formula is known and the algorithm terminates. If the candidate is not a winning strategy, then the opposing player synthesizes a *counter-strategy* to beat it. In the next round, the opposing player proposes a new strategy that beats all previous strategies, and the two players switch roles. The process continues until one of the players obtains a winning strategy.

The next section defines the terminology and notation used in the rest of the paper. In § 3 we define *strategy skeletons*. Strategy skeletons are similar to classical strategies, but have an order structure: our LRA decision procedure searches for winning strategy skeletons by ascending in this ("improvement") order. § 4 describes a procedure for constructing a *counter-strategy* to a candidate strategy skeleton. § 5 presents the decision procedure for LRA. § 6 discusses how to extend the strategy improvement procedure to other theories, in particular linear integer arithmetic. We present experimental results in § 7 and conclude in § 8.

2 Game Semantics for Linear Arithmetic

The definitions that follow are mostly standard, but the reader who is unfamiliar with the game-theoretic interpretation of quantifiers in first-order logic may wish to read § 2.2.

2.1 Linear Arithmetic

The syntax of linear rational arithmetic (LRA) is as follows. The set of terms is defined by the following grammar

$$s, t \in \mathsf{Term} ::= c \mid x \mid s + t \mid c \cdot t$$

where x is a variable symbol and c is a rational number. Ground formulas are defined by the grammar

$$F, G \in \mathsf{Formula} ::= t < 0 \mid t = 0 \mid F \land G \mid F \lor G$$

Notice that we (without loss of generality) assume that formulas are negation-free. A *prenex formula* is a formula of the form

$$\varphi = \mathcal{Q}_1 x_1.\mathcal{Q}_2 x_2.\cdots \mathcal{Q}_n x_n.F\ ,$$

where each $\mathcal{Q}_i$ is either $\exists$ or $\forall$, F is a ground formula, and all variable symbols $\{x_1, ..., x_n\}$ are assumed to be distinct. For a formula φ, we use $\mathsf{fv}(\varphi)$ to denote the free variables which appear in φ; similarly, $\mathsf{fv}(t)$ denotes the free variables of the term t. A prenex formula is a *sentence* if it has no free variables.

A *valuation* is a function $M : V \to \mathbb{Q}$, where V is some finite set of variable symbols and $\mathbb{Q}$ denotes the set of rational numbers. For a term t and a valuation V, we use $[\![t]\!]^M$ to denote the interpretation of t within the valuation M. We use $M \models \varphi$ to denote that M satisfies the formula φ (M is a model of φ), defined in the usual way. Many modern SMT solvers have the capability of computing satisfying valuations for satisfiable ground formulas [De Moura and Bjørner, 2008; Barrett *et al.*, 2011; Dutertre, 2014].

For a valuation M, a variable x, and a rational number c, we use $M\{x \mapsto c\}$ to denote the extension of M where x is interpreted as c:

$$M\{x \mapsto c\} \triangleq \lambda y.\text{if } y = x \text{ then } c \text{ else } M(y)$$

For a formula φ, variable x, and term t, we use $\varphi[x \mapsto t]$ to denote the formula obtained from φ by substituting each free occurrence of x with t. For a sequence π, we use $|\pi|$ to denote the length of π and π_i to denote the i^{th} element of π.

2.2 Satisfiability Games

A prenex sentence

$$\varphi = \mathcal{Q}_1 x_1.\mathcal{Q}_2 x_2.\cdots \mathcal{Q}_n x_n.F$$

defines a *satisfiability game*, which is played as follows. There are two players, SAT and UNSAT, which take turns picking rational numbers. At round i of the game, if $\mathcal{Q}_i$ is $\exists$, then SAT chooses a rational number to assign to the variable x_i; if $\mathcal{Q}_i$ is $\forall$, then the choice belongs to UNSAT. After playing this game for n rounds, the players' choices define a *play* $\rho \in \mathbb{Q}^n$: a sequence of rational numbers of length n. This play can be identified with a valuation of the variables $M^\rho : \{x_1, ..., x_n\} \to \mathbb{Q}$ where for each i, $M^\rho(x_i) \triangleq \rho_i$. The SAT player wins ρ if $M^\rho \models F$, otherwise UNSAT wins.

A *strategy* for a satisfiability game determines the next move for a player as a function of the sequence of previous moves in the game:

Definition 2.1 (Strategy)**.** *Let*

$$\varphi = \mathcal{Q}_1 x_1.\mathcal{Q}_2 x_2.\cdots \mathcal{Q}_n x_n.F$$

be a prenex LRA sentence. A SAT strategy *for the satisfiability game φ is a function*

$$f : \{\rho \in \mathbb{Q}^* : |\rho| < n \land \mathcal{Q}_{|\rho|+1} = \exists\} \to \mathbb{Q}$$

Similarly, an UNSAT strategy *for φ is a function*

$$g : \{\rho \in \mathbb{Q}^* : |\rho| < n \land \mathcal{Q}_{|\rho|+1} = \forall\} \to \mathbb{Q}$$

We say that a play ρ of φ *conforms* to a SAT strategy f if for every $i \in \{1, ..., n\}$ such that $\mathcal{Q}_i$ is $\exists$, we have

$$\rho_i = f(\rho_1 ... \rho_{i-1})\ .$$

That is, $\rho_i = f(\rho_1 ... \rho_{i-1})$ whenever $f(\rho_1 ... \rho_{i-1})$ is defined. Similarly, a play ρ of φ conforms to an UNSAT strategy g if $\rho_i = g(\rho_1 ... \rho_{i-1})$ whenever $g(\rho_1 ... \rho_{i-1})$ is defined.

We say that a SAT strategy f is *winning* if SAT wins every play that conforms to f. Similarly, an UNSAT strategy g is winning if UNSAT wins every play that conforms to g. It is easy to show that φ is satisfiable if and only if the SAT player has a winning strategy of the satisfiability game for φ (and φ is unsatisfiable if and only if the UNSAT player has a winning strategy).

For any prenex sentence φ, we use $\neg\varphi$ to refer to the negation-free formula equivalent to the negation of φ, obtained in the usual way. The formula $\neg\varphi$ defines a *dual game*, which is played as φ but with the roles of the SAT and UNSAT player reversed. It is often useful to define terminology and algorithms for the SAT player and leave the analogous definition for the UNSAT player implicit by appealing to duality. For example, rather than defining UNSAT strategies explicitly, we could define an UNSAT strategy to be a SAT strategy for the dual game $\neg\varphi$. Note that, due to the completeness of the theory of linear rational arithmetic, we have:

Proposition 2.2. *Let φ be a prenex sentence. UNSAT has a winning strategy for φ if and only if SAT has a winning strategy for the dual game $\neg\varphi$.*

3 Strategy Skeletons

As defined in § 2.2, a strategy is a function that determines the next move for a player starting from any position in the game. A strategy *skeleton* determines a finite *set* of possible moves for any position. A key feature of strategy skeletons (which are formally defined below) is that they are ordered: one skeleton is "better" than another if it associates more moves with every position. This order will be exploited by the strategy improvement algorithm in § 5, that operates by proposing a sequence of increasingly "better" candidate strategy skeletons.

Definition 3.1 (Strategy Skeleton)**.** *Let*

$$\varphi = \mathcal{Q}_1 x_1.\mathcal{Q}_2 x_2.\cdots \mathcal{Q}_n x_n.F$$

be a prenex LRA sentence. A SAT strategy skeleton *for φ is a finite, non-empty set $S \subseteq (\mathsf{Term} \cup \{\bullet\})^n$ of sequences over terms plus a distinguished placeholder $\bullet$, where each* *sequence $\pi_1 \cdots \pi_n \in S$ has length n and such that for all $i \in \{1, ..., n\}$,*

- *if $\mathcal{Q}_i$ is $\exists$, then π_i is a term and $\mathsf{fv}(\pi_i) \subseteq \{x_1, ..., x_{i-1}\}$*
- *if $\mathcal{Q}_i$ is $\forall$, then π_i is $\bullet$*

An UNSAT strategy skeleton for φ is defined to be a SAT strategy skeleton for the dual game $\neg\varphi$.

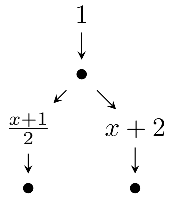
Example 3.2. *Consider the following formula:*

$$\varphi \triangleq \exists w.\forall x.\exists y.\forall z.(y < 1 \lor 2w < y) \land (z < y \lor x < z)$$

One possible SAT strategy skeleton for φ is:

$$\{1\bullet((x+1)/2)\bullet, 1\bullet(x+2)\bullet\}\ ,$$

which is visualized as the tree to the right. This tree indicates the moves available to a SAT player who plays according to this skeleton: on turn 1, the SAT player must choose 1. On turn 2, the choice belongs to the UNSAT player (represented by the placeholder $\bullet$). On turn 3, SAT may choose between $(x+1)/2$ and $x+2$ (where x is the value the UNSAT player chose in the previous turn). Turn 4 again belongs to the UNSAT player, after which the game is finished.



Similar to the way that a strategy can be interpreted as a collection of plays (the plays that conform to that strategy), a strategy skeleton can be interpreted as a collection of strategies. We make this interpretation precise by defining what it means for a strategy to conform to a skeleton. Let $\varphi = \mathcal{Q}_1 x_1.\mathcal{Q}_2 x_2.\cdots \mathcal{Q}_n x_n.F$ be a prenex LRA sentence, and let S be a strategy skeleton for the SAT player on φ. We say that a play ρ of φ *conforms* to S if there exists some $\pi_1 \cdots \pi_n \in S$ such that for all $i \in \{1, ..., n\}$ such that $\mathcal{Q}_i$ is $\exists$, we have $[\![x_i]\!]^{M^\rho} = [\![\pi_i]\!]^{M^\rho}$. We say that a strategy f for φ conforms to S if every play that conforms to f also conforms to S. A strategy skeleton is *winning* if some winning strategy conforms to it.

Winning formulas The goal of our decision procedure for LRA is to compute a winning skeleton for one of the players. The next step in developing this algorithm is to give a method for answering the question *is a given candidate strategy skeleton winning*? This question can be encoded into a universally quantified formula (the *winning formula* for the skeleton) which is satisfiable if and only if the skeleton is winning. The intuition behind this encoding is that we may replace each existential quantifier in the formula (representing the infinitely many possible moves available to the SAT player) with a finite disjunction (representing the finitely many possible moves that are available to the SAT if the play must conform to the given skeleton).

Formally, we define the winning formula $\mathsf{win}(S, \varphi)$ for a strategy skeleton S for the game φ recursively as follows:

$$\mathsf{win}(S, \exists x_i.\psi) \triangleq \bigvee \{\mathsf{win}(S', \psi)[x_i \mapsto t] : S \to^t S'\}$$
$$\mathsf{win}(S, \forall x_i.\psi) \triangleq \forall x_i.\mathsf{win}(\{\pi : \bullet\pi \in S\}, \psi)$$
$$\mathsf{win}(S, F) \triangleq F$$

where we write $S \to^t S'$ iff $S' = \{\pi : t\pi \in S\}$ and S' is non-empty.

Proposition 3.3. *Let φ be a formula and let S be a SAT strategy skeleton for φ. There is a winning SAT strategy for the game φ that conforms to S if and only if $\mathsf{win}(S, \varphi)$ is satisfiable.*

Example 3.4. *Again consider the formula φ and SAT strategy skeleton from Example 3.2. The winning formula is*

$$\forall x.(\forall z.(\frac{x+1}{2} < 1 \lor 2 < \frac{x+1}{2}) \land (z < \frac{x+1}{2} \lor x < z))$$
$$\lor(\forall z.(x+2 < 1 \lor 2 < x+2) \land (z < x+2 \lor x < z))$$

One may check that the winning formula is satisfiable, and therefore the strategy skeleton is winning and φ is satisfiable.

4 Counter-strategies

If a given strategy skeleton is *not* winning, then the opposing player has a counter-strategy skeleton that beats it (that is, the counter-strategy wins against every strategy that conforms to the given skeleton). In this section, we formalize counter-strategies and give an algorithm for synthesizing them.

Given a formula φ, a SAT strategy skeleton S for φ, and an UNSAT strategy skeleton U for φ, we say that U is a *counter-strategy* for S (*U beats S*) if there exists a strategy f that conforms to U such that every play that conforms to both S and f is a win for UNSAT. Counter-strategies for UNSAT strategy skeletons are defined similarly. We may observe the following:

Observation 4.1 (Anti-symmetry)**.** *Let S be a SAT strategy skeleton and U be an UNSAT strategy skeleton. It cannot be the case that S is a counter-strategy for U and U is a counter-strategy for S.*

This kind of anti-symmetry is the key to our strategy improvement algorithm making progress. Throughout the course of the algorithm, SAT will propose a sequence of strategies $S_0, S_1, ...$ and UNSAT will propose a sequence of strategies $U_0, U_1, ...$, that are arranged as follows:

$$\begin{tikzcd}
S_0 & \subseteq & S_1 \arrow[dll] & \subseteq & S_2 \arrow[dll] & \cdots \\
U_0 \arrow[u, "beats"] & \subseteq & U_1 \arrow[u] & \subseteq & U_2 \arrow[u] & \cdots
\end{tikzcd} \quad (\dagger)$$

The inclusions $S_0 \subseteq S_1 \subseteq ...$ (and $U_0 \subseteq U_1 \subseteq ...$) hold by construction: S_{i+1} is defined to be the union of S_i and a counter-strategy that beats U_i. Anti-symmetry ensures that the inclusions are strict, so that the players make progress towards a winning strategy.

We now consider the question of how counter-strategies may be synthesized. Given a formula φ and a SAT strategy S, Algorithm 1 either finds a counter-strategy to S or determines that no counter-strategy exists (that is, S is a winning strategy skeleton). (By duality, passing Algorithm 1 the formula $\neg\varphi$ and an UNSAT strategy U for φ finds a counter-strategy to U or determines that U is winning).

We explain Algorithm 1 by illustrating its operation on the formula from Example 3.2,

$$\varphi \triangleq \exists w.\forall x.\exists y.\forall z.(y < 1 \lor 2w < y) \land (z < y \lor x < z)$$

```
1  Procedure has-counter-strategy($(S, \varphi)$)
       Input  : LRA sentence $\varphi = \mathcal{Q}_1 x_1 \cdots \mathcal{Q}_n x_n.F$,
                SAT strategy $S$
       Output: Counter-strategy to $S$ if one exists;
                None if no counter-strategy exists
       /* Compute Herbrandized winning formula */
2      for each $\pi$ such that $\pi \bullet \pi' \in S$ for some $\pi'$ do
3          $herbrand[\pi\bullet] \leftarrow$ fresh Herbrand constant
4      $win \leftarrow false$
5      for $\pi \in S$ do
6          $G \leftarrow F$
7          for $i \leftarrow n$ downto $1$ do
8              if $\mathcal{Q}_i$ is $\exists$ then
9                  $G \leftarrow G[x_i \mapsto \pi_i]$
10             else
11                 $G \leftarrow G[x_i \mapsto herbrand[\pi_1 \cdots \pi_i]]$
12         $win \leftarrow win \vee G$
       /* win is the Herbrandized winning formula */
13     if $\neg win$ is satisfiable then
           /* Synthesize a counter-strategy for $S$ */
14         Let $M \models \neg win$
15         $(U, G) \leftarrow \mathsf{css}(\varphi, M, \lambda x.\bot, \epsilon, S)$      /* Alg. 2 */
16         return Counter-strategy $U$
17     else
           /* $S$ is a winning strategy */
18         return None
```

Algorithm 1: Check if a strategy has a counter-strategy

using the SAT strategy $S = \{0\bullet x\bullet, 0\bullet(2x)\bullet\}$ for φ. Following the definition of win from the previous section, the winning formula for S is as follows:

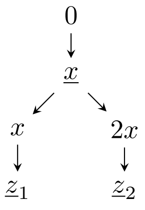
$$\forall x.(\forall z.(x < 1 \vee 0 < x) \wedge (z < x \vee x < z)) \\ \vee(\forall z.(2x < 1 \vee 0 < 2x) \wedge (z < 2x \vee x < z))$$

Algorithm 1 begins by computing a Herbrandization of this winning formula (replacing each universally quantified variable with a fresh constant symbol), so that witnesses for each quantified variable can be computed from a model of its negation (should a model exist). The auxiliary map *herbrand* keeps track of the symbols introduced by Herbrandization:

$$herbrand[0\bullet] = \underline{x}$$
$$herbrand[0\bullet x\bullet] = \underline{z}_1$$
$$herbrand[0\bullet(2x)\bullet] = \underline{z}_2$$

After lines 2-12, the Herbrandized winning formula *win* is:

$$win = ((\underline{x} < 1 \vee 0 < \underline{x}) \wedge (\underline{z}_1 < \underline{x} \vee \underline{x} < \underline{z}_1)) \\ \vee((2\underline{x} < 1 \vee 0 < 2\underline{x}) \wedge (\underline{z}_2 < 2\underline{x} \vee \underline{x} < \underline{z}_2))$$



Notice that there is only one Herbrand constant ($\underline{x}$) corresponding to the variable x and there are two ($\underline{z}_1$ and $\underline{z}_2$) corresponding to the variable z. The intuition behind this can be illustrated by the structure of the satisfiability game tree for φ when the SAT player conforms to S, depicted to the right. The SAT player begins the game by playing 0 for w. The UNSAT player responds by choosing a value for x (which we call $\underline{x}$). The SAT player then has a choice to play either the same value that UNSAT played or twice that value. For each of these two moves, the UNSAT player may choose a different value for z: $\underline{z}_1$ corresponds to UNSAT's choice in the first case, and $\underline{z}_2$ corresponds to UNSAT's choice in the second.

```
Procedure css($(\varphi, M, M^\pi, \pi, S)$)
    Input  : LRA formula $\varphi = \mathcal{Q}x_i \cdots \mathcal{Q}x_n.F$,
             Valuation $M$ : $Image(herbrand) \to \mathbb{Q}$
             Valuation $M^\pi : \{x_1, ..., x_{i-1}\} \to \mathbb{Q}$
             Path $\pi \in (\mathsf{Term} \cup \{\bullet\})^{i-1}$
             SAT strategy $S$ for $\varphi$
    Output: $(U, G)$, where
             $U$ is an UNSAT strategy
             $G$ is a formula, and such that
             $M^\pi \models G$, and
             For any $M' \models G$, $U$ beats $S$ starting from $M'$
    if $i > n$ then
        return $(\{\epsilon\}, \neg F)$
    $\varphi' \leftarrow \mathcal{Q}_{i+1} x_{i+1} \cdots \mathcal{Q}_n x_n.F$
    if $\mathcal{Q}_i$ is $\forall$ then
        $M^{\pi\bullet} \leftarrow M^\pi\{x_i \mapsto [\![herbrand[\pi\bullet]]\!]^M\}$
        $S' \leftarrow \{\pi' : \bullet\pi' \in S\}$
        $(U, G) \leftarrow \mathsf{css}(\varphi', M, M^{\pi\bullet}, \pi\bullet, S')$
        $\overline{t} \leftarrow \mathsf{select}(M^{\pi\bullet}, x_i, G)$
        return $(\{\overline{t}\overline{\pi} : \overline{\pi} \in U\}, G[x_i \mapsto \overline{t}])$
    else
        $U \leftarrow \emptyset$
        $G \leftarrow true$
        for $S \to^t S'$ do
            $M^{\pi t} \leftarrow M^\pi\{x_i \mapsto [\![t]\!]^{M^\pi}\}$
            $(U^+, G^+) \leftarrow \mathsf{css}(\varphi', M, M^{\pi t}, \pi t, S')$
            $G \leftarrow G \wedge (G^+[x_i \mapsto t])$
            $U \leftarrow U \cup U^+$
        return $(\{\bullet\overline{\pi} : \overline{\pi} \in U\}, G)$
```

Algorithm 2: Counter-strategy synthesis

After computing the Herbrandized winning formula *win*, Algorithm 1 checks if $\neg win$ is satisfiable using an SMT solver. If $\neg win$ is unsatisfiable then the procedure returns: *win* is satisfiable and so S is a winning strategy skeleton (by Proposition 3.3). Otherwise, the SMT solver returns a model of $\neg win$, say

$$M = \{\underline{x} \mapsto -2, \underline{z}_1 \mapsto -2, \underline{z}_2 \mapsto -3\}$$

M corresponds to an UNSAT strategy that beats S:

$$g(\rho) = \begin{cases} -2 & \text{if } |\rho| = 1 \\ -2 & \text{if } |\rho| = 3 \wedge \rho_2 = \rho_3 \quad (\textit{left path}) \\ -3 & \text{otherwise} \quad (\textit{right path}) \end{cases}$$

The next step of Algorithm 1 is to use the model M to synthesize a counter-strategy *skeleton* for S that generalizes the strategy g, using Algorithm 2. Algorithm 2 traverses the satisfiability game tree pictured above: on the way **down** (traversing a path π from the root to a leaf), it builds a valuation (M^π) representing the unique play of the game where SAT conforms to π and UNSAT conforms to g. Since g beats S, this play is a win for UNSAT (i.e., $M^\pi \models \neg((y < 1 \vee 2w <$

$y) \wedge (z < y \vee x < z)))$. For example, given the model M above the two paths in the example give the models:

$$\textit{Left: } \{w \mapsto 0, x \mapsto -2, y \mapsto -2, z \mapsto -2\}$$
$$\textit{Right: } \{w \mapsto 0, x \mapsto -2, y \mapsto -4, z \mapsto -3\}$$

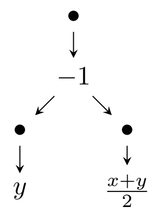
When Algorithm 2 moves **up** the tree, it builds a counter-strategy skeleton U and a formula G such that:

(i) The model corresponding to the *unfinished* play π is a model of G (i.e., $M^\pi \models G$), and

(ii) U beats S starting from any play ρ such that $M^\rho \models G$.

Thus, at a recursive call of Algorithm 2 of depth i, U is a counter-strategy skeleton for S playing the part of the game *after* the i^{th} quantifier (leaving the first i moves fixed as in M^π), and G is a formula that constrains the moves *before* the i^{th} quantifier.

If depth i corresponds to an existential quantifier, then the counter-strategy is extended by prepending the placeholder value (line 18). If depth i corresponds to a universal quantifier in φ, then Algorithm 2 uses a *model-guided term selection function* select to select an appropriate term with which to extend the counter-strategy (making conditions (i) and (ii) hold). § 4.1 gives an implementation of select.



From property (ii) of Algorithm 2, we can conclude that when the call $\mathsf{css}(\varphi, M, \lambda x.\bot, \epsilon, S)$ terminates, it returns a pair (U, G) where U is a counter-strategy for S on the game φ, and G is *true*. The final counter-strategy U that is synthesized by Algorithm 2 on this example is pictured to the right.

Finally, we summarize the preceding discussion in the following proposition:

Proposition 4.2. *Let φ be a formula and let S be a strategy skeleton for φ. If S is a winning strategy for φ, then* $\mathsf{has\text{-}counter\text{-}strategy}(S, \varphi)$ *returns* None. *If S is not winning, then* $\mathsf{has\text{-}counter\text{-}strategy}(S, \varphi)$ *returns a strategy skeleton for the* UNSAT *player on φ that beats S.*

4.1 Model-guided term selection

This sub-section defines select, the model-guided term selection procedure used in Algorithm 2. This function is inspired by model-based projection, an under-approximate quantifier elimination technique proposed in [Komuravelli *et al.*, 2014], and the decision procedure for LRA presented in [Weispfenning, 1988] (a modification of the one proposed in [Ferrante and Rackoff, 1975]).

Given a ground formula F, a model $M \models F$, and a variable $x \in \mathsf{fv}(F)$, the model-guided term selection function $\mathsf{select}(M, x, F)$ must find a term t such that x does not appear in t ($\mathsf{fv}(t) \subseteq \mathsf{fv}(F) \setminus \{x\}$) and $M \models F[x \mapsto t]$.

Observe that every atomic proposition in F that contains x can be written (after re-writing using standard arithmetical rules) in one of three forms: $x = s$, $x < s$, or $s < x$ (where x does not appear in s). Let $\mathit{EQ}(M, x, F)$ be the set of all terms s such that $x = s$ appears in F and $[\![x]\!]^M = [\![s]\!]^M$, let $\mathit{UB}(M, x, F)$ be the set of all terms s such that $x < s$ appears in F and $[\![x]\!]^M < [\![s]\!]^M$, and let $\mathit{LB}(M, x, F)$ be the set of all terms s such that $s < x$ appears in F and $[\![s]\!]^M < [\![x]\!]^M$. Since (by assumption) M is a model of F, M is also a model of $F[x \mapsto t]$ for any term t such that t satisfies the same equations, lower bounds, and upper bounds as x (i.e., $[\![t]\!]^M = [\![s]\!]^M$ for all $s \in \mathit{EQ}(M, x, F)$, $[\![t]\!]^M < [\![s]\!]^M$ for all $s \in \mathit{UB}(M, x, F)$, and $[\![s]\!]^M < [\![t]\!]^M$ for all $s \in \mathit{LB}(M, x, F)$).

The procedure $\mathsf{select}(M, x, F)$ proceeds as follows. If $\mathit{EQ}(M, x, F)$ is non-empty, then we define $\mathit{eq}(M, x, F)$ to be some arbitrarily-chosen member. Otherwise, suppose that $\mathit{UB}(M, x, F)$ is non-empty. Then there exists a (not necessarily unique) least upper bound $u \in \mathit{UB}(M, x, F)$ such that $[\![u]\!]^M \leq [\![s]\!]^M$ for all $s \in \mathit{UB}(M, x, F)$. We let $\mathit{lub}(M, x, F)$ be an arbitrarily-chosen least upper bound for x if one exists. Similarly, we define $\mathit{glb}(M, x, F)$ to be an arbitrarily-chosen greatest lower bound if one exists. Finally, we define select: $\mathsf{select}(M, x, F) \triangleq$

$$\begin{cases} \mathit{eq}(M, x, F) & \text{if } \mathit{EQ}(M, x, F) \neq \emptyset \\ \frac{1}{2}(\mathit{lub}(M, x, F) + \mathit{glb}(M, x, F)) & \begin{array}{l}\text{if } \mathit{UB}(M, x, F) \neq \emptyset \\ \text{and } \mathit{LB}(M, x, F) \neq \emptyset\end{array} \\ \mathit{lub}(M, x, F) - 1 & \text{if } \mathit{UB}(M, x, F) \neq \emptyset \\ \mathit{glb}(M, x, F) + 1 & \text{if } \mathit{LB}(M, x, F) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

The term $\mathsf{select}(M, x, F)$ satisfies the same equations, lower bounds, and upper bounds as x. As a result, we have the following lemma, which is sufficient for the correctness argument of Algorithm 2 (in particular, the lemma is sufficient to prove that the algorithm maintains properties (i) and (ii) above):

Lemma 4.3 (Model preservation)**.** *Suppose $M \models F$. Then $M \models F[x \mapsto \mathsf{select}(M, x, F)]$.*

The function select also satisfies a *finite-image* property, which is crucial for the termination argument of the decision procedure for LRA that we present in the next section:

Lemma 4.4 (Finite Image)**.** *Let F be a formula and x be a variable. The set $\{\mathsf{select}(M, x, F) : M \models F\}$ is finite.*

Proof. Define an equivalence relation $\equiv_F$ on valuations, where $M \equiv_F M'$ if and only if M and M' satisfy the set of same atomic propositions in F. There are finitely many equivalence classes of $\equiv_F$ (since there are finitely many atomic propositions in F), and select selects equal terms for equivalent models, and so the set $\{\mathsf{select}(M, x, F) : M \models F\}$ is finite. □

5 A strategy improvement algorithm for LRA

This section describes a decision procedure for linear rational arithmetic based on strategy improvement. The algorithm is given in Algorithm 3. Given an input formula

$$\varphi = \mathcal{Q}_1 x_1 \cdots \mathcal{Q}_n x_n . F$$

the algorithm operates as follows. First, query an SMT solver for a model of the formula F. If no model exists, then clearly φ is unsatisfiable. If a model does exist, then we may use it to construct an initial strategy skeleton for the SAT player, similarly to the way that Algorithm 2 constructs a counter-strategy from a model (lines 4-13).

```
Input  : LRA sentence $\varphi = \mathcal{Q}_1 x_1 \cdots \mathcal{Q}_n x_n.F$
Output: true if $\varphi$ is satisfiable, false if $\varphi$ is unsatisfiable
if $F$ is unsatisfiable then
    return false
/* Compute initial strategy for SAT */
Let $M \models F$
$G \leftarrow F$
for $i \leftarrow n$ downto $1$ do
    if $\mathcal{Q}_i$ is $\exists$ then
        $t \leftarrow \mathsf{select}(M, x_i, G)$
        $G \leftarrow G[x_i \mapsto t]$
        $\pi_i \leftarrow t$
    else
        $G \leftarrow G[x_i \mapsto \mathsf{select}(M, x_i, G)]$
        $\pi_i \leftarrow \bullet$
$S \leftarrow \{\pi_1 \cdots \pi_n\}$
$U \leftarrow \emptyset$
/* Strategy improvement */
while true do
    switch has-counter-strategy$(S, \varphi)$ do
        case Counter-strategy $U'$
            $U \leftarrow U \cup U'$
        otherwise
            /* No counter strategy $\Rightarrow$ $S$ is winning */
            return true
    switch has-counter-strategy$(U, \neg\varphi)$ do
        case Counter strategy $S'$
            $S \leftarrow S \cup S'$
        otherwise
            return false
```

Algorithm 3: Satisfiability modulo LRA

After constructing the initial strategy skeleton to SAT, we begin the strategy improvement phase of the algorithm, depicted in Diagram †. At the start of the loop, we have a SAT strategy skeleton S and an UNSAT strategy skeleton U, such that S beats U (or U is empty). First, we try to synthesize a counter-strategy to S. If counter-strategy synthesis fails, the algorithm terminates: S is a winning strategy, so φ is satisfiable. If has-counter-strategy does synthesize a counter-strategy, it is added to the candidate UNSAT strategy U (*improving* it). Next, we repeat this process for the candidate UNSAT strategy U, and either terminate upon proving that U is a winning strategy or find a counter-strategy with which to improve S and continue looping.

Algorithm 3 returns *true* only when it has synthesized a SAT strategy skeleton for which the winning formula is satisfiable (i.e., the negation of its winning formula is unsatisfiable), and returns *false* only when it has synthesized an UNSAT strategy skeleton for which the winning formula is satisfiable. Thus, partial correctness of Algorithm 3 is an immediate corollary of Proposition 3.3: if Algorithm 3 returns *true*, then φ is satisfiable, and if Algorithm 3 returns *false*, then φ is unsatisfiable.

The termination argument for Algorithm 3 is based on two properties: *progress* (as the algorithm progresses, the strategy skeleton S is strictly increasing), and *finiteness* (there is a finite bound on the size of S). The progress argument (mentioned in § 4) comes from the anti-symmetry property of counter-strategies (Observation 4.1). The finiteness argument is by induction on quantifier depth. The base case is by Lemma 4.4. If we assume that the finiteness condition holds for all strategies for a given formula φ of depth n, then we prove the same holds for depth $n + 1$ by arguing that the algorithm computes the first move of the game by calling $\mathsf{select}(M, x, F)$, where F is some formula obtained by substituting terms from a counter-strategy of depth n into a negated, Hebrandized winning formula of strategy of depth n, M is a model of F, and x is the variable associated with the first move of the game. By the induction hypothesis, the set of all possible such F is finite, so by Lemma 4.4, the set of terms that could be selected for x is finite, and thus the set of skeletons is finite.

Combining the above arguments for partial correctness and termination, we close the section with the following theorem:

Theorem 5.1. *Algorithm 3 is a decision procedure for LRA.*

6 Beyond LRA

The focus of this article is satisfiability in the theory of linear rational arithmetic, but the core ideas behind the strategy improvement algorithm can be extended to other theories. This section discusses what we require of a theory in order to apply our algorithm.

There are three assumptions on the theory that must be met in order to use strategy improvement as a decision procedure.

1. The quantifier-free fragment of the theory in question must be decidable, and models for satisfiable formulas must be effectively constructable.
2. The theory must be complete. This is required because Algorithm 3 checks that the (universally quantified) winning formula for a skeleton is satisfiable by checking that its (existentially quantified) negation is unsatisfiable.
3. The theory admits a model-guided term selection function that is model-preserving (Lemma 4.3) and has finite images (Lemma 4.4).

Thus, the design work involved in extending the strategy improvement algorithm to a new theory is in devising a term selection function. In fact, condition 3 can be weakened slightly to require only that a theory admits a model-guided *virtual* term selection function. A virtual term is a term that does not belong to the theory in question, but which may be evaluated in any model of the theory and for which substitution is theory-definable. Section 6.1 gives a term selection function (based on ideas from [Cooper, 1972; Komuravelli *et al.*, 2014]) for linear integer arithmetic that makes use of such virtual terms.

Remark 6.1. *It is worth noting that our requirements for virtual terms are stronger than ones usually employed by quantifier elimination procedures [Cooper, 1972; Loos and Weispfenning, 1993; Komuravelli* et al.*, 2014; Bjørner and Janota, 2015]. For example, the LIA quantifier elimination procedure from [Cooper, 1972] makes use of the virtual term ∞, which does not meet our requirements because it cannot be evaluated in the standard model of the integers.*

We leave the development of term selection functions for other theories as a promising direction for future work.

6.1 Term selection for Linear Integer Arithmetic

Linear integer arithmetic is an important theory for application in program analysis and verification. The syntax of LIA is the same as for LRA, except that constants are integers, and the language is extended with divisibility predicates $a|t$ (where $a \in \mathbb{Z}$ is a positive integer and t is a term). In the remainder of the section we will develop a virtual term selection function for linear integer arithmetic.

First we define virtual terms and virtual substitution. We consider virtual terms of the form $\lfloor t/a \rfloor + b$, where t is a term with $\mathsf{fv}(t) \subseteq \mathsf{fv}(F) \setminus \{x\}$, $a \in \mathbb{Z}$ is a positive integer, and $b \in \mathbb{Z}$ is an integer. The syntax of LIA does not admit integral division $\lfloor t/a \rfloor$, so virtual terms do not belong to the syntax of LIA. However, integral division can be interpreted in any valuation ($\llbracket \lfloor t/a \rfloor \rrbracket^M$ is well-defined), and for any formula φ, variable x, and virtual term $\lfloor t/a \rfloor + b$, we can perform a *virtual* substitution that yields a formula equivalent to $\varphi[x \mapsto \lfloor t/a \rfloor + b]$ but which belongs to the syntax of LIA:

Definition 6.2 (Virtual Substitution)**.** *Let φ be a formula, x be a variable, and $\lfloor t/a \rfloor + b$ be a virtual term. Without loss of generality, we assume that every atomic proposition in φ in which x appears takes one of the following forms:*

$$cx < s \qquad\qquad s < cx \qquad\qquad d|cx + s$$

where s denotes a term with $x \notin \mathsf{fv}(s)$ and c and d denote positive integers. (Note that the equality symbol is not needed because $s = s'$ may be replaced by $s < s' + 1 \land s' < s + 1$.)

We define the virtual subsitution $\varphi[x \rightsquigarrow \lfloor t/a \rfloor + b]$ *to be the formula obtained by renaming the bound variables in φ to avoid capture and replacing each atomic proposition in φ in which x appears as follows:*

$$cx < s \mapsto \bigvee_{i=0}^{a-1} a|(t-i) \land c(t - i + ab) < as$$

$$s < cx \mapsto \bigvee_{i=0}^{a-1} a|(t-i) \land as < c(t - i + ab)$$

$$d|cx + s \mapsto \bigvee_{i=0}^{a-1} a|(t-i) \land ad|c(t - i + ab) + as$$

The development of the previous three sections can be repeated with virtual terms and virtual subsitution in place of terms and classical substitution. The only thing that remains to extend the strategy improvement algorithm to LIA is to define a virtual term selection function $\mathsf{vselect}$ that satisfies the model preservation and finite image properties.

Let F be a LIA formula, let $M \models F$ be a model, and let x be a variable. We require $\mathsf{vselect}(M, x, F)$ to satisfy the model preservation property

$$M \models F[x \rightsquigarrow \mathsf{vselect}(M, x, F)] \ .$$

Since F is negation-free, it is sufficient for model preservation to hold on all the atomic propositions of F that contain x. Observe that every atomic propsition in F that contains x can be written in one of three forms: $(cx < s)$, $(s < cx)$, or $(d|cx + s)$ (where x does not appear in s). Let $\mathit{UB}(M, x, F)$ be the set of all atoms $(cx < s)$ that appear in F such that $c\llbracket x \rrbracket^M < \llbracket s \rrbracket^M$, let $\mathit{LB}(M, x, F)$ be the set of all atoms $(s < cx)$ that appear in F such that $\llbracket s \rrbracket^M < c\llbracket x \rrbracket^M$, and let $\mathit{Div}(M, x, F)$ be the set of all divisibility atoms $(d|cx + s)$ in F such that d divides $\llbracket cx + s \rrbracket^M$.

First we consider the constraints that the divisibility atoms $\mathit{Div}(M, x, F)$ place on $\mathsf{vselect}(M, x, F)$. Let $(d|cx + s) \in \mathit{Div}(M, x, F)$. Observe that for any integer $z \in \mathbb{Z}$,

$$\begin{aligned} d|cz + \llbracket s \rrbracket^M &\iff cz \equiv c\llbracket x \rrbracket^M \mod d \\ &\iff z \equiv \llbracket x \rrbracket^M \mod (d/\gcd(|c|, d)) \end{aligned}$$

Thus, the divisibility atom $(d|cx + s)$ is satisfied so long as

$$\llbracket \mathsf{vselect}(M, x, F) \rrbracket^M \equiv \llbracket x \rrbracket^M \mod (d/\gcd(|c|, d)) \ .$$

To collect all such divisibility constraints into one, we define

$$\Delta(M, x, F) \triangleq \mathrm{lcm}\{\tfrac{d}{\gcd(|c|,d)} : (d|cx + s) \in \mathit{Div}(M, x, F)\}$$

and require that

$$\llbracket \mathsf{vselect}(M, x, F) \rrbracket^M \equiv \llbracket x \rrbracket^M \mod \Delta(M, x, F) \ .$$

Next, we consider the constraints that the upper bound atoms $\mathit{UB}(M, x, F)$ place on $\mathsf{vselect}(M, x, F)$. Suppose that $\mathit{UB}(M, x, F)$ is non-empty. Then there exists a term t, a positive integer a, and non-negative integer b less than $\Delta(M, x, F)$ such that:

1. $(ax < t) \in \mathit{UB}(M, x, F)$,
2. $\llbracket \lfloor (t-1)/a \rfloor - b \rrbracket^M \equiv \llbracket x \rrbracket^M \mod \Delta(M, x, F)$, and
3. For any other t', a', b' with the above properties, we have $\llbracket \lfloor (t-1)/a \rfloor - b \rrbracket^M \leq \llbracket \lfloor (t'-1)/a' \rfloor - b' \rrbracket^M$.

We define $\mathit{lub}(M, x, F)$ to be a virtual term $\lfloor (t-1)/a \rfloor - b$ satisfying these three properties (picking one arbitrarily if there are several choices). Property 2 ensures that $\mathit{lub}(M, x, F)$ satisfies all divisibility constraints in $\mathit{Div}(M, x, F)$ and property 3 ensures that $\mathit{lub}(M, x, F)$ satisfies all upper bound constraints $\mathit{UB}(M, x, F)$. Property 2 and the fact that b is less than $\Delta(M, x, F)$ implies that $\llbracket x \rrbracket^M \leq \llbracket \mathit{lub}(M, x, F) \rrbracket^M$, and so $\mathit{lub}(M, x, F)$ also satisfies all lower bound constraints. However, if $\mathit{UB}(M, x, F)$ is empty, we need to consider lower bound constraints explicitly, so we define $\mathit{glb}(M, x, F)$ analogously to $\mathit{glb}(M, x, F)$ if $\mathit{LB}(M, x, F)$ is non-empty. Finally, we define $\mathsf{vselect}$:

$$\mathsf{vselect}(M, x, F) \triangleq \begin{cases} \mathit{lub}(M, x, F) & \text{if } \mathit{UB}(M, x, F) \neq \emptyset \\ \mathit{glb}(M, x, F) & \text{if } \mathit{LB}(M, x, F) \neq \emptyset \\ \llbracket x \rrbracket^M \mod \Delta(M, x, F) & \text{otherwise} \end{cases}$$

The argument that the finite image property holds for $\mathsf{vselect}$ is the same as the one for select (Lemma 4.4).

7 Experimental Evaluation

We have implemented Algorithm 3 in a prototype tool called SimSat. The tool is implemented in OCaml and uses Z3 to solve ground formulas [De Moura and Bjørner, 2008].

Comparison with related techniques

Heuristic quantifier instantiation is a sound but incomplete technique that is commonly used to handle quantifiers in SMT solvers [De Moura and Bjørner, 2007; Ge *et al.*, 2007]. Our

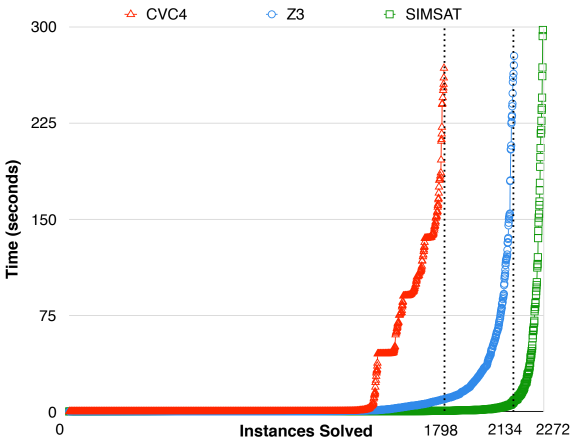


Figure 1: Distribution of run-time over solved instances

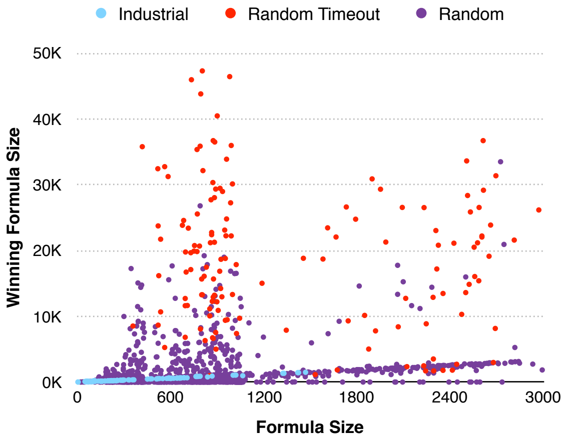


Figure 2: Formula size vs. Winning formula size

experimental evaluation compares with the experimental configuration of CVC4 [Barrett *et al.*, 2011], which won the LRA category in the 2015 SMT competition. CVC4 uses a portfolio of quantifier instantiation techniques.

Bjørner and Janota recently developed a decision procedure for LRA (as well as other theories) that is based on the intuition of satisfiability games [Bjørner and Janota, 2015]. Conceptually, their procedure solves satisfiability games by exploring the game tree in a *forwards* direction. The SAT and UNSAT player take turns instantiating quantifiers until one of them loses, and then backjumps to an earlier quantification level and learns a blocking clause to remove a part of the search space that will result in a loss for that player. In contrast, in Algorithm 3, players take turns synthesizing strategies for the entire game, rather than synthesizing the next move. Algorithm 3 requires solving larger formulas (corresponding to the whole game), but the payoff is a more "global" perspective of the game.

Dutertre developed an efficient algorithm for solving $\exists^*\forall^*$ in the theory of linear rational arithmetic [Dutertre, 2015]. At a high level, Dutertre's algorithm operates similarly to Algorithm 3 when restricted to the $\exists^*\forall^*$ fragment. Dutertre uses a term selection function similar to the one in § 4.1, but with some interesing heuristic improvements (that do not extend to the case of arbitrary quantification in an obvious way).

Monniaux developed a lazy quantifier elimination algorithm for LRA formulas with alternating quantifiers that is based on geometric quantifier elimination (polyhedra projection) [Monniaux, 2010]. This algorithm was implemented in a satisfiability procedure in Z3 [Phan *et al.*, 2012]. The experimental evaluation in [Bjørner and Janota, 2015] shows that Bjørner and Janota's algorithm outperforms lazy quantifier elimination, so we omit it from our evaluation.

Results

We evaluated SimSat on a suite of benchmarks drawn from SMT-LIB2 [Barrett *et al.*, 2010] and Mjollnir [Monniaux, 2010]. The experimental evaluation was performed on a Linux machine with Intel Core i5 2.80GHz processors and 4GB of memory. The time limit was set to 300 seconds.

The table below summarizes the number of solved problem instances by each tool. The results are divided into three categories: industrial benchmarks (from SMT-LIB2) with an $\exists^*\forall^*$ quantifier prefix, industrial benchmarks (from SMT-LIB2) with a non-$\exists^*\forall^*$ quantifier prefix (all of which happened to have a quantifier prefix of the form $\exists^*\forall\exists$), and random benchmarks (from both SMT-LIB2 and Mjollnir). SimSat, Z3 (implementing the algorithm from [Bjørner and Janota, 2015]), CVC4, and Yices (implementing the algorithm from [Dutertre, 2015]) all solve all industrial $\exists^*\forall^*$ benchmarks (all tools have a mean running time of less than 0.01 seconds). On the remaining industrial benchmarks, SimSat and Z3 solve all instances (SimSat mean time 1 second, Z3 mean time 0.02 seconds) while CVC4 solves 83%. On the random benchmarks, SimSat dominates (93%), followed by Z3 (86%) and CVC4 (71%).

	SimSat	Z3	CVC4	Yices
Industrial $\exists^*\forall^*$ (247)	**247**	**247**	**247**	**247**
Industrial $\exists^*\forall\exists$ (144)	**144**	**144**	119	–
Random (2030)	**1881**	1743	1432	–

The distribution of running times of the three tools across random benchmarks is depicted in the cactus plot in Figure 1 (a point (x, y) in the plot represents that x instances are solved in y seconds). Note that SimSat can solve in ∼5.3s as many instances as Z3 can solve in 300s.

Figure 2 plots the size of input formulas against the size of the winning formula for the winning strategy computed by SimSat (or the last candidate strategy if SimSat did not terminate within 300 seconds). Formula size is measured as the number of nodes in a DAG representation of the formula. For legibility, the plot truncates input formula size at 3000 and the winning formula size at 50000. Note that on the industrial benchmarks, the relationship between input formula size and winning formula size is linear.

Linear integer arithmetic We also evaluated SimSat as a decision proecedure for linear integer arithmetic, using the virtual term selection procedure described in § 6. The benchmarks are drawn from SMT-LIB2 and randomly generated benchmarks. The table below summarizes the number of solved problem instances by each tool (excluding Yices, which does not implement an LIA solver). SimSat and Z3 both solve all industrial instances (SimSat mean time 1.2 seconds, Z3 mean time 0.1 seconds), while CVC4 solves 59%. On the random benchmarks, SimSat solves the most instances (71%), followed by CVC4 (70%) and Z3 (58%).

	SimSat	Z3	CVC4
Industrial (390)	**390**	**390**	231
Random (300)	**212**	174	211

8 Conclusion

This article presents a decision procedure for the theory of linear arithmetic based on strategy improvement for satisfiability games. There are several avenues for future work in this direction. The strategy improvement algorithm is very sensitive to model selection, so it would be interesting to experiment with heuristics for different models of ground formulas. Another promising direction is to extend the strategy synthesis algorithm to other decidable theories, such as the theory of algebraic data types. Another direction is to investigate uses for the strategy synthesis capability of the algorithm: just as there are many applications for models of ground formulas, we believe there may be interesting uses for winning strategies of quantified formulas.

References

[Barrett *et al.*, 2010] Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.

[Barrett *et al.*, 2011] Clark Barrett, Christopher L Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *CAV*, pages 171–177, 2011.

[Bjørner and Janota, 2015] Nikolaj Bjørner and Mikolas Janota. Playing with quantified satisfaction. In *LPAR*, 2015.

[Cooper, 1972] David C Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 7(91-99), 1972.

[De Moura and Bjørner, 2007] Leonardo De Moura and Nikolaj Bjørner. Efficient E-matching for SMT solvers. In *CADE*, pages 183–198. 2007.

[De Moura and Bjørner, 2008] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.

[Dutertre, 2014] Bruno Dutertre. In *CAV*, pages 737–744, 2014.

[Dutertre, 2015] Bruno Dutertre. Solving exists/forall problems with Yices. In *Workshop on Satisfiability Modulo Theories*, 2015.

[Ferrante and Rackoff, 1975] Jeanne Ferrante and Charles Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM Journal on Computing*, 4(1):69–76, 1975.

[Ge *et al.*, 2007] Yeting Ge, Clark Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. In *CADE*, pages 167–182. 2007.

[Ghilardi and Ranise, 2010] Silvio Ghilardi and Silvio Ranise. MCMT: A model checker modulo theories. In *Automated Reasoning*, pages 22–29. 2010.

[Hintikka, 1982] Jaakko Hintikka. Game-theoretical semantics: insights and prospects. *Notre Dame Journal of Formal Logic Notre-Dame, Ind.*, 23(2):219–241, 1982.

[Janota *et al.*, 2012] Mikoláŝ Janota, William Klieber, Joao Marques-Silva, and Edmund Clarke. Solving QBF with counterexample guided refinement. In *SAT*, pages 114–128. 2012.

[Komuravelli *et al.*, 2014] Anvesh Komuravelli, Arie Gurfinkel, and Sagar Chaki. SMT-based model checking for recursive programs. In *CAV*, pages 17–34, 2014.

[Kovács and Voronkov, 2013] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In *CAV*, pages 1–35, 2013.

[Loos and Weispfenning, 1993] Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993.

[Monniaux, 2010] David Monniaux. Quantifier elimination by lazy model enumeration. In *CAV*, pages 585–599, 2010.

[Phan *et al.*, 2012] Anh-Dung Phan, Nikolaj Bjørner, and David Monniaux. Anatomy of alternating quantifier satisfiability (work in progress). In *Workshop on Satisfiability Modulo Theories*, page 6, 2012.

[Reynolds *et al.*, 2015] Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli, and Clark Barrett. Counterexample-guided quantifier instantiation for synthesis in SMT. In *CAV*, pages 198–216. 2015.

[Schulz, 2013] Stephan Schulz. System Description: E 1.8. In *LPAR*, pages 735–743, 2013.

[Solar-Lezama *et al.*, 2006] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. In *ASPLOS*, pages 404–415, 2006.

[Solar-Lezama, 2008] Armando Solar-Lezama. *Program synthesis by sketching*. PhD thesis, University of California, Berkeley, 2008.

[Weispfenning, 1988] Volker Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5(1):3–27, 1988.

[Zhang, 2006] Lintao Zhang. Solving QBF with combined conjunctive and disjunctive normal form. In *AAAI*, 2006.