Quantified Linear Arithmetic Satisfiability Via Fine-grained Strategy Improvement

Omitted for reviewing

Abstract. Checking satisfiability of formulae in the theory of linear arithmetic has far reaching applications, including program verification and synthesis. Many satisfiability solvers excel at proving and disproving satisfiability of quantifier-free linear arithmetic formulas and have recently begun to support quantified formulas. Beyond simply checking satisfiability of formulas, fine-grained strategies for satisfiability games enables solving additional program verification and synthesis tasks. Quantified Satisfiability games are played between two players—SAT and UNSAT—who take turns instantiating quantifiers and choosing branches of boolean connectives to evaluate the given formula. A winning strategy for SAT (resp. UNSAT) determines the choices of SAT (resp. UNSAT) as a function of UNSAT's (resp. SAT's) choices that results in the given formula evaluating to true (resp. false) no matter what choices UNSAT (resp. SAT) may make. As we are interested in both checking satisfiability *and* synthesizing winning strategies, we must avoid conversion to normal-forms that alter the game semantics of the formula (e.g. prenex normal form). We present fine-grained strategy improvement and strategy synthesis, the first technique capable of synthesizing winning fine-grained strategies for linear arithmetic satisfiability games, which may be used in higher-level applications. We experimentally evaluate our technique against existing quantified satisfiability solvers and find it performs favorably against state-of-the-art solvers.

1 Introduction

Checking satisfiability of quantified formulae in the theory of linear (integer or real) arithmetic (LA) has applications to a broad class of problems (e.g., program verification and synthesis). Satisfiability modulo theory (SMT) solvers excel at deciding satisfiability of the ground (quantifier free) fragment of first order theories (e.g., LA). Other techniques like first order theorem solvers work well for quantified formulae but have limited support for theories. Typically, SMT solvers either perform quantifier elimination, which is often computationally expensive, or try to heuristically instantiate quantifiers, which is sound but incomplete for deciding satisfiability.

Recently, decision procedures have been developed to check satisfiability of quantified LA formulae directly [4,8,16,5]. Both Bjørner and Janota [4]'s and Farzan and Kincaid [8]'s decision procedures require that the input fomula be in prenex normal form—the formula is a sequence of quantifiers followed by a quantifier free formula. Checking satisfiability of a formula in prenex normal

form is undesirable for two reasons: (1) conversion to prenex normal form may increase the number of quantifier alternations within the formula and (2) conversion to prenex normal form may change the semantics of the formula—the scope of a quantifier may include new irrelevant quantified variables. An increase in the number of quantifier alternations makes checking satisfiability more computationally costly. For many applications, a simple yes or no is insufficient (e.g. program synthesis, angelic symbolic execution, or invariant generation), instead a strategy determining how the formula was proven Sat or Unsat is required. No existing technique can produce fine-grained strategies. Either they do not produce strategies [4,1,5], or convert the formula to prenex normal form [8], thus altering the game semantics—e.g., when used to synthesize a term used in one path, conversion to prenex normal form may allow finding a strategy that uses variables only defined on a different program path.

This paper presents a decision procedure for checking satisfiability of quantified LA formulae. This paper generalizes the *coarse-grained* strategy improvement—strategies for quantifier—to a class of *fine-grained* strategies—strategies for both quantifiers and connectives.

Our procedure is inspired by the game semantics for quantifiers and connectives [12]. Every LA formula induces a game between two players, SAT and UNSAT. SAT tries to prove the formula satisfiable, while UNSAT tries to prove it unsatisfiable. The players take turns instantiating quantifiers or choosing a sub-formula of boolean connectives. SAT controls existential quantifiers and disjunctions, while UNSAT controls universal quantifiers and conjunctions. SAT wins the game if the chosen model satisfies the chosen sub-formula; otherwise, UNSAT wins. A LA formula is satisfiable exactly when SAT has a winning strategy to the induced game—SAT can win no matter what choices UNSAT makes.

Fine-grained strategy improvement exploits the fine-grained structure of LA formulas (quantifiers and boolean connectives) to formulate a recursive procedure that iteratively improves a candidate strategy via computing winning strategies to induced sub-games. We generalize the notion of strategies and counter-strategy computation from Farzan and Kincaid [8] to handle quantifiers and connectives as well as allowing computing counter-strategies with a fixed-prefix (to enable the recursive nature of fine-grained strategy improvement). Fine-grained strategy improvement improves upon existing techniques by (1) avoiding conversion to prenex normal form or (2) allow extracting a proof object (a winning strategy) that determines exactly how the formula is proven (un)satisfiable.

For simplicity, the remainder of this paper provides details for linear rational arithmetic (LRA); however, the algorithmic details and game semantics provided in this paper are directly applicable to linear integer arithmetic (LIA). In section 2, we review the game semantics for linear arithmetic [12], and its relation with LRA satisfiability. Sections 3, 5, and 4 present the procedure to compute winning strategy skeletons, whose existence proves or disproves LRA satisfiability. Section 6 shows how to compute a winning strategy from a strategy skeleton. Sections 7 and 8 compares this work to others. The Appendix contains implementation details, proofs, and extended experimental results.

2 Fine-grained Game Semantics for LRA Satisfiability

This section reviews the syntax of **Linear Rational Arithmetic** (LRA) and its game semantics. Both are run-of-the-mill, Section 2.1 describes the syntax or LRA considered and Section 2.2 the corresponding game semantics.

2.1 Linear Rational Arithmetic

The syntax of LRA is formed from two sets—Terms and Formulas. The grammar for terms and formulae parameterized over a set of variables X is as follows:

$$s, t \in \mathbf{Term}(X) ::= c \in \mathbb{Q} \mid x \in X \mid s + t \mid c \cdot t$$
$$\varphi, \psi \in \mathbf{Formula}(X) ::= t < 0 \mid t = 0 \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \forall x.\ \varphi \mid \exists x.\ \varphi$$

Without loss of generality, this paper considers negation free formulae and assumes that every variable bound by a quantifier within a formula to be distinct. For a formula φ, $FV(\varphi)$ denotes its free variables. Similarly, $FV(t)$ denotes the free variables of term t. A **sentence** is a LRA formula with no free variables. A **ground formula** is a quantifier-free formula which may contain free variables. $\mathbf{Term}(V)$ and $\mathbf{Formula}(V)$ denotes the set of terms and formulae, respectively, whose free variables belong to the set of variables V.

A **valuation**, $M \in V \to \mathbb{Q}$, is a function mapping from a finite set of variables, $V \subseteq X$, to the rationals. We use $[\![t]\!]^M$ to denote the value of t within the valuation M—assuming $FV(t) \subseteq dom(M)$—with the usual interpretation. $M \models \varphi$ denotes that M satisfies the formula φ (we say M is a model of φ).

For a valuation M, a variable x, and a rational constant c, $M\{x \mapsto c\}$ denotes the valuation M except with x mapped to 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, $\varphi[x \mapsto t]$ represents the formula obtained by substituting every free occurrence of x with t.

2.2 Fine-grained Game Semantics

For a more thorough introduction, Hintikka describes the game semantics for FOL formulae [12]. Every LRA sentence defines a *satisfiability game*, which is played between two players: SAT and UNSAT. The players take turns choosing instantiations for quantifiers and sub-formulae of connectives. SAT controls the choices for existential quantifiers and disjunctions, while UNSAT controls universal quantifiers and conjunctions.

Formally, a state of a LRA-*satisfiability game* for a LRA-sentence φ is $G(\psi, M)$, where ψ is a sub-formula of φ and M is a valuation. The initial state of the satisfiability game for φ is $G(\varphi, \emptyset)$. Below gives the rules of the game with the

assumption that $FV(\psi) \subseteq dom(M)$.

$G(t < 0, M)$ SAT wins if $M \models t < 0$. Otherwise, UNSAT wins.
$G(t = 0, M)$ SAT wins if $M \models t = 0$. Otherwise, UNSAT wins.
$G(\varphi \wedge \psi, M)$ UNSAT chooses to either play $G(\varphi, M)$ or $G(\psi, M)$.
$G(\varphi \vee \psi, M)$ SAT chooses to either play $G(\varphi, M)$ or $G(\psi, M)$.
$G(\forall x.\varphi, M)$ UNSAT picks $c \in \mathbb{Q}$ and then plays $G(\varphi, M\{x \mapsto c\})$.
$G(\exists x.\varphi, M)$ SAT picks $c \in \mathbb{Q}$ and then plays $G(\varphi, M\{x \mapsto c\})$.

A *strategy* for SAT or UNSAT determines that player's next move as a function of all the moves previously played. In the above definition of a LRA-satisfiability game, the state $G(\psi, M)$ implicitly represents the moves made so far. This is made explicit by representing a *play* of the game as a sequence of rational numbers (instantiating quantifiers) and the labels L and R (choosing the left or right branch of a disjunction or conjunction). For the formula φ and play π, we represent the sub-formula and valuation forming the state of the game after playing π as φ^π and m^π, respectively. Both are defined as follows:

$$\varphi^\epsilon \triangleq \varphi \quad (\forall x.\varphi)^{c\cdot\pi} \triangleq \varphi^\pi \quad (\exists x.\varphi)^{c\cdot\pi} \triangleq \varphi^\pi$$
$$M^\epsilon \triangleq \emptyset \quad M^{c\cdot\pi} \triangleq M^\pi\{x \mapsto c\} \quad M^{c\cdot\pi} \triangleq M^\pi\{x \mapsto c\}$$

$$(\varphi \wedge \psi)^{L\cdot\pi} \triangleq \varphi \quad (\varphi \wedge \psi)^{R\cdot\pi} \triangleq \psi \quad (\varphi \vee \psi)^{L\cdot\pi} \triangleq \varphi \quad (\varphi \vee \psi)^{R\cdot\pi} \triangleq \psi$$
$$M^{L\cdot\pi} \triangleq M^\pi \quad M^{R\cdot\pi} \triangleq M^\pi \quad M^{L\cdot\pi} \triangleq M^\pi \quad M^{R\cdot\pi} \triangleq M^\pi$$

If φ^π does not evaluate using the above rules, then π is an *illegal* play and φ^π is undefined. In the remainder of this paper, we use "play" to mean "legal play." A play π is **complete** when φ^π is an atom (neither player has any move to make). For any complete play π, SAT *wins* if and only if $M^\pi \models \varphi^\pi$. Similarly, UNSAT wins if and only if $M^\pi \not\models \varphi^\pi$.

For any formula φ, $\neg\varphi$ denotes the negation-free formula equivalent to the negation of φ. The sentence $\neg\varphi$, induces the dual satisfiability game of φ – a game played in the same manner as φ but with the roles of SAT and UNSAT swapped. This duality is used to define terminology and algorithms explicitly for SAT and implicity for UNSAT as the corresponding SAT version for $\neg\varphi$.

Definition 1 (Strategy). *Let $\mathcal{M} = \mathbb{Q} \cup \{L, R\}$ be the set of all moves, $f : \mathcal{M}^* \to \mathcal{M}$ be a partial function from sequences of moves to a move, and π a sequence of moves. The play π **conforms** to f exactly when $\pi_i = f(\pi_1, \ldots, \pi_{i-1})$ whenever $f(\pi_1, \ldots, \pi_{i-1})$ is defined.*

*Let φ be a LRA-stentence, a SAT **strategy** for φ is a partial function $f : M^* \to M$, which has the property that for any play π that conforms to f, (1) if φ^π is $F \vee G$ then $f(\pi)$ is defined and $f(\pi) \in \{L, R\}$ and (2) if φ^π is $\exists x.F$ then $f(\pi)$ is defined and $f(\pi) \in \mathbb{Q}$.*

The SAT strategy f is **winning** if every complete play that conforms to f is won by SAT. It is well-known that φ is satisfiable if and only if SAT has a winning strategy.

3 Fine-grained Strategy Skeletons

This section defines fine-grained SAT strategy skeletons that form the basis of our fine-grained strategy improvement algorithm (cf. Algorithm 1). A SAT strategy skeleton is an abstraction that represents multiple possible strategies that SAT may choose. Recall that in Section 2.2, we defined strategies to be a function that maps a play of a satisfiability game to the next move of the game. A *strategy skeleton* similarly maps a play of the satisfiability game to a finite set of *possible* moves to play next. At a high-level, the strategy improvement algorithm iteratively finds better and better strategy skeletons via the computation of counter-strategy skeletons (cf. Section 5).

Example 1. To illustrate fine-grained strategy skeletons and the algoritms presented in this paper consider the formula φ which forms the basis of the running example we use throughout this paper:

$$\varphi \triangleq \forall x, z.\ (x = z \vee (\exists y.\ (x < y \wedge y < z) \vee (z < y \wedge y < x)))$$

The formula φ expresses the fact that for any pair of rational numbers x and z, either x and z are equal or there is some value y between x and z. To the right, we display a SAT strategy skeleton for φ which we call S. The two $\bullet$ symbols act as placeholders for the values chosen by UNSAT for the quantified variables x and z. The skeleton encodes that no matter what values ($\bar{x}$ and $\bar{z}$) UNSAT chooses to instantiate x and y with, SAT chooses to play the the left branch of the disjunction leading to the atom $x = z$—at the end of the path we display $\bar{x} = \bar{z}$ this atom after substituting the placeholder values for UNSAT's choice for the formally bound variables.

As seen in Example 1, SAT skeletons are tree-like structures that follow the structure of φ. Formally, SAT strategy skeletons for a LRA-satisfiability game φ, are represented as a set of paths. We use $\mathit{Skel}(\varphi, \mathit{vars})$ to denote the set of SAT strategy skeletons for φ whose terms may range over the set of variables *vars*. For a sub-skeleton of a sentence, *vars* represents the set of quantifiers that φ is in scope of. The set of strategy skeletons for a sentence is thus $\mathit{SKEL}(\varphi, \emptyset)$. For a set of paths S, $\ell \cdot S = \{\ell \cdot \pi : \pi \in S\}$ denotes the set obtained by prepending each path in S with the label ℓ. A skeleton is a subset of $(\text{TERM} \cup \{\bullet, L, R\})^*$ (whose specific form depends on the formula φ). We define *SKEL* as the least solution to the following set of rules:

$$\frac{\varphi \text{ is atomic}}{\{\epsilon\} \in \mathit{SKEL}(\varphi, \mathit{vars})} \qquad \frac{S \in \mathit{SKEL}(\varphi, \mathit{vars})}{L \cdot S \in \mathit{SKEL}(\varphi \vee \psi, \mathit{vars})} \qquad \frac{S \in \mathit{SKEL}(\psi, \mathit{vars})}{R \cdot S \in \mathit{SKEL}(\varphi \vee \psi, \mathit{vars})}$$

$$\frac{S \in \mathit{SKEL}(\varphi, \mathit{vars}) \qquad T \in \mathit{SKEL}(\psi, \mathit{vars})}{(L \cdot S) \cup (R \cdot T) \in \mathit{SKEL}(\varphi \wedge \psi, \mathit{vars})} \qquad \frac{S \in \mathit{SKEL}(\varphi, \mathit{vars} \cup \{x\})}{\bullet \cdot S \in \mathit{SKEL}(\forall x.\ \varphi, \mathit{vars})}$$

$$\frac{t \in \text{TERM}(\mathit{vars}) \qquad S \in \mathit{SKEL}(\varphi, \mathit{vars} \cup \{x\})}{(t \cdot S) \in \mathit{SKEL}(\exists x.\ \varphi, \mathit{vars})} \qquad \frac{S, T \in \mathit{SKEL}(\varphi, \mathit{vars})}{(S \cup T) \in \mathit{SKEL}(\varphi, \mathit{vars})}$$

Just as strategies can be thought of as a collection of plays, strategy skeletons can be thought of as a collection of strategies. Similar to strategies and plays, we can determine when a strategy **conforms** to a strategy skeleton. We say a SAT strategy f conforms to a strategy skeleton S when every complete play π conforming to f conforms to S. A play π conforms to S, if there is some path $\rho \in S$ such that $|\pi| = |\rho|$ and for each i we have (1) $\varphi^{\pi_0,\ldots,\pi_{i-1}} = \exists x.\psi$ for some ψ and $[\![x]\!]^{M^\pi} = [\![\rho_i]\!]^{M^\pi}$, or (2) $\varphi^{\pi_0,\ldots,\pi_{i-1}}$ is a disjunctive or conjunctive formula and $\pi_i = \rho_i$, or (3) $\varphi^{\pi_0,\ldots,\pi_{i-1}}$ is a universally quantified formula and $\rho_i = \bullet$. A strategy skeleton is **winning** if there is *a* winning strategy that conforms to it.

In order to develop a decision procedure that produces a winning strategy skeleton, we first turn to the problem of determining if a SAT skeleton S for the LRA satisfiability game φ is winning. Given S and φ the winning formula of S is a universally quantified sentence that is satisfiable exactly when S is a winning SAT skeleton for φ. Rather than directly checking the winning formula, we check if the formula $\text{lose}(S, \varphi)$—a formula equivalent to the negation of the winning formula—is unsatisfiable. This formulation is preferable for two reasons: (1) the formula is existentially quantified and can be easily skolemized to a quantifier free formula and checked with an off-the-shelf SMT solver and (2) as shown in Section 5 a model of the skolemized formula can be used to construct an UNSAT strategy for φ that beats S. We define $\text{lose}(S, \varphi)$ as follows:

$$
\begin{aligned}
\text{lose}(\emptyset, \varphi) &\triangleq \mathit{true} \\
\text{lose}(\{\epsilon\}, \varphi) &\triangleq \neg\varphi \\
\text{lose}(S, \varphi \vee \psi) &\triangleq \text{lose}(\{\pi : L \cdot \pi \in S\}, \varphi) \wedge \text{lose}(\{\pi : R \cdot \pi \in S\}, \psi) \\
\text{lose}(S, \varphi \wedge \psi) &\triangleq \text{lose}(\{\pi : L \cdot \pi \in S\}, \varphi) \vee \text{lose}(\{\pi : R \cdot \pi \in S\}, \psi) \\
\text{lose}(S, \exists x.\varphi) &\triangleq \bigwedge \{\text{lose}(S', \varphi)[x \mapsto t] : S' = \{\pi : t \cdot \pi \in S\}\} \\
\text{lose}(S, \forall x.\varphi) &\triangleq \exists x.\text{lose}(\{\pi : \bullet \cdot \pi \in S\}, \varphi)
\end{aligned}
$$

Proposition 1. *Let φ be a LRA sentence and S a SAT strategy skeleton for φ. S is winning if and only if lose(S, φ) is unsatisfiable.*

4 Fine-grained Strategy Improvement

This section presents a novel strategy improvement algorithm, Algorithm 1, for deciding LRA satisfiability. The algorithm recursively decomposes the overall game into a series of smaller games. Algorithm 1 assumes that UNSAT makes the first move in the LRA-satisfiability game φ. If SAT would instead play first, Algorithm 1 may be applied to $\neg\varphi$ and the result negated. The first step of the strategy improvement algorithm initializes a SAT skeleton. Any SAT skeleton $S \in \mathit{SKEL}(\varphi, \emptyset)$ may be used.

After initialization, the algorithm will check if a counter-strategy exists (cf. Section 5)—an UNSAT skeleton such that some strategy conforming to it beats all strategies conforming to SAT's skeleton. If there is no counter-strategy, then

Algorithm 1: Satisfiability modulo LRA

Function Solve(φ,M^π,S)
  Input: LRA Formula $\varphi = \psi^\pi$ for some sentence ψ.
  Valuation $M^\pi : (x_0, \ldots, x_n) \to \mathbb{Q}$ such that $FV(\varphi) \subseteq dom(M^\pi)$.
  S a SAT skeleton for φ.
  switch has-counter-strategy(S, M^π, φ) **do**
    case *Counter-strategy* U **do**
      $\pi', U' \leftarrow peel(\varphi, U)$;
      switch Solve($\neg\varphi^{\pi'}$, $M^\pi \cup M^{\pi'}$, U') **do**
        case *Sat* U'' **do**
          return *Unsat* $\{\pi'\pi'' : \pi'' \in U''\}$
        case *Unsat* S' **do**
          return Solve(φ, M^π, $S \cup \{(\neg\pi') \cdot \pi : \pi \in S'\}$
    case *default* **do**
      return *Sat* S

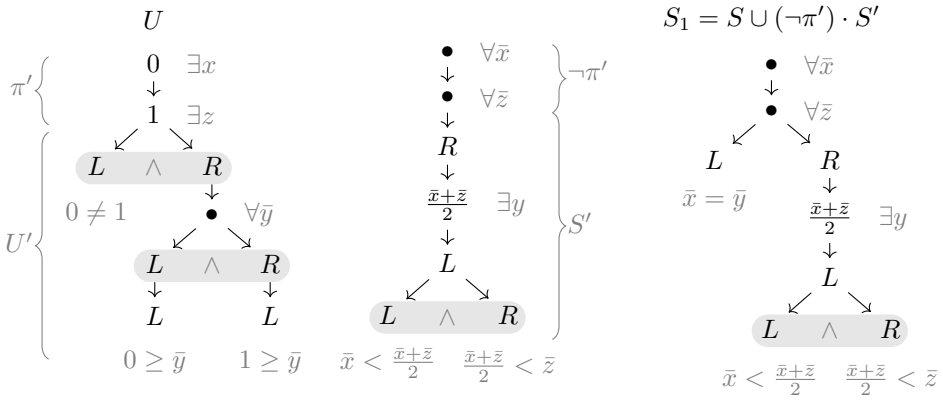
Function Strategy-Improvement(φ)
  Let $S \in SKEL(\varphi, \emptyset)$ be any skeleton for φ;
  switch Solve(φ, $\lambda x.\ \bot$, S) **do**
    case *Sat* S' **do**
      return *true*
    case *Unsat* U **do**
      return *false*

SAT has a winning strategy conforming to it's skeleton. Otherwise, UNSAT has a counter-strategy U. The auxillary function *peel* uses φ and U to compute π'—the leading universal and conjunctive moves—and U'—the remaining skeleton (i.e. $U = \{\pi'\pi : \pi \in U'\}$). The algorithm continues by fixing the moves in π' and having the players swap places while solving the resulting sub-game $\neg\varphi^{\pi'}$. Formally, *peel* is defined as follows:

$$
\begin{aligned}
peel(\forall x.F, U) &\triangleq \langle t \cdot \pi, U' \rangle && \text{where } \langle \pi, U' \rangle = peel(F, U'') \text{ and } U = \{t \cdot \pi : \pi \in U''\} \\
peel(F \wedge G, U) &\triangleq \langle L \cdot \pi, U' \rangle && \text{where } \langle \pi, U' \rangle = peel(F, U'') \text{ and } U = \{L \cdot \pi : \pi \in U''\} \\
peel(F \wedge G, U) &\triangleq \langle R \cdot \pi, U' \rangle && \text{where } \langle \pi, U' \rangle = peel(G, U'') \text{ and } U = \{R \cdot \pi : \pi \in U''\} \\
peel(\varphi, U) &\triangleq \langle \epsilon, U \rangle && \text{otherwise}
\end{aligned}
$$

By construction, the leading SAT moves of a counter-strategy must form a single path—Algorithm 3 only chooses a single term or disjunct when constructing a counter-strategy. This ensures that *peel* is properly defined. After peeling off the leading existential and disjunctive moves from U, the algorithm recursively solves the resulting sub-game. Example 2 continues the running example.

Example 2.

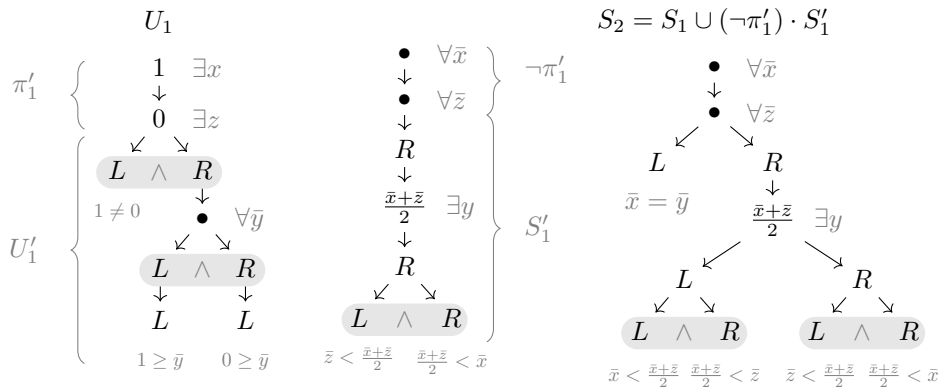


After using *peel* to construct π' and U', Algorithm 1 recursively solves the sub-game $\neg\varphi^{\pi'} \triangleq x \neq z \wedge \forall y.\, (x \geq y) \vee (y \geq z) \wedge (z \geq y) \vee (y \geq x))$ starting with U' and the model $M^{\pi'} = \{x \mapsto 0, z \mapsto 1\}$. The sub-game is played with the role of the two players switched—the recursive call uses the formula $\neg\varphi^{\pi'}$ rather than $\varphi^{\pi'}$—thus, U' is a SAT skeleton for the resulting sub-game *and* the assumption that the top-level connective of φ is controlled by UNSAT is maintained.

Either SAT or UNSAT have a winning skeleton to $G(\neg\varphi^{\pi'}, M^{\pi'})$. If $G(\neg\varphi^{\pi'}, M^{\pi'})$ is won by SAT, then UNSAT wins $G(\varphi^{\pi}, M^{\pi})$ with the UNSAT skeleton U''. Thus, the algorithm may terminate and return UNSAT's winning skeleton U''. Otherwise, $G(\neg\varphi^{\pi'}, M^{\pi'})$ is won by UNSAT. This is the case depicted in Example 2. The recursive call returns the winning skeleton S', displayed in the center of Example 2. The skeleton S' will instantiate y with the average of x and z and chose the left disjunct $x < y < z$, which clearly beats U' when x is 0 and z is 1. The sub-skeleton S' can be extended to counter U by prepending every path of S' with the "negation" of π' the initial moves of UNSAT. The path π' may be negated by converting each term in π' to a $\bullet$—i.e. $(\neg\pi')_i = \pi'_i$ if $\pi'_i \in \{L, R\}$, otherwise $(\neg\pi')_i = \bullet$. Technically, $(\neg\pi') \cdot S'$ is not a skeleton when π' contains disjunctive moves—the negation is a conjunctive branch and the resulting set of paths only covers one of the branches, while a skeleton must cover both branches—however, when unioned with S the initial skeleton for SAT, the final result S_1 is a skeleton and is a counter-skeleton to U. S_1 is depicted on the right side of Example 2.

While S_1 counters U, it is not yet winning. SAT will lose any play where unsat instantiates x and z such that $z < x$. Example 3 continues this example. On the next iteration of the game, the algorithm finds U_1 a counter-skeleton to S_1. Just as before the procedure splits apart U_1 and solves the induced sub-game. The procedure finds that SAT wins the sub-game with the skeleton S'_1. The new skeleton is extended and combined with S_1 to form S_2 and the current game φ continues starting from S_2; however, on the next iteration, the procedure determines that S_2 has no counter-strategy and is thus a winning SAT skeleton for the game φ.

Example 3.



Theorem 1. *Algorithm 1 is a decision procedure for LRA satisfiability.*

5 Counter-strategies

When a strategy skeleton is not winning—its losing formula is satisfiable—the opposing player must have a counter-strategy that beats every strategy that conforms to the strategy skeleton. Given a model of the losing formula, this section shows how to construct such a counter-strategy skeleton.

Section 4 presents an algorithm that produces a winning strategy skeleton by recursively computing winning strategies to sub-games. The key idea is that the current player fixes their initial move and synthesizes a winning strategy to the subgame. The algorithm returns to the current game where either the current player won the sub-game and their initial choice is also winning, or the opposing player won the sub-game and the current player will try to find a new initial move that counters the strategy skeleton of the opposing player. If no such initial move is possible, then the opposing player wins the current game.

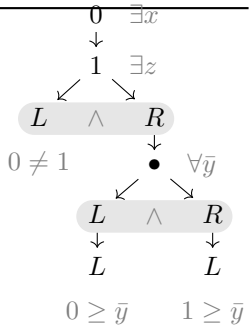
Counter-strategies form the basis of our approach. Fix a LRA-satisfiability game φ, play π of φ, SAT skeleton S for φ^π, and UNSAT skeleton U for φ^π. U is a **counter-strategy** of S (U *beats* S), if there is some strategy g conforming to U such that for every strategy f conforming to S, UNSAT wins every complete play $\pi\pi'$ such that π' conforms to both f and g. By definition, it cannot be the case that U beats S and S beats U. This asymmetry is essential to ensuring that the strategy improvement algorithm makes progress towards verifying or falsifying the formula φ.

Example 4. Recall the initial strategy S from Example 1, in which SAT always chose the branch with the atom $x = z$ no matter what values UNSAT chose for x and z. The losing formula of S is $lose(S) \triangleq \bar{x} \neq \bar{z}$ which summarizes the choices of $\bar{x}$ and $\bar{z}$ that UNSAT may make to falsify the atom $x = z$ SAT choose. The losing formula of S is satisfiable—e.g., with the model $M = \{\bar{x} \mapsto 0, \bar{z} \mapsto 1\}$.

Algorithm 2: Check if a given strategy skeleton has a counter-strategy

Function has-counter-strategy(S, M_0, φ)

Input: LRA formula φ.
Valuation $M_0 : (x_0, \ldots, x_n) \to \mathbb{Q}$ s.t. $FV(\varphi) \subseteq dom(M_0)$
S a strategy skeleton for φ
foreach π *such that* $\pi \bullet \pi' \in S$ *for some* π' **do**
  $H[\pi\bullet] \leftarrow$ fresh rational variable
foreach π *such that* $\pi L \pi' \in S$ *for some* π' *and* φ^π *is a conjunction* **do**
  $H[\pi L] \leftarrow$ fresh boolean variable
  $H[\pi R] \leftarrow$ fresh boolean variable
lose $\leftarrow$ *true*
foreach $\pi \in S$ **do**
  win $\leftarrow \varphi^\pi\{x \mapsto M_0(x) : x \in dom(M)\}$
  conds $\leftarrow$ *true*
  for $i \leftarrow |\pi|$ *to* 1 **do**
    $\pi' \leftarrow \pi_1, \ldots, \pi_{i-1}$
    if $\varphi^{\pi'} = F \wedge G$ **then**
      conds $\leftarrow$ *conds* $\wedge$ (*win* $\Rightarrow H[\pi' \cdot \pi_i]$)
      win $\leftarrow H[\pi' \cdot L] \wedge H[\pi' \cdot R]$
    else if $\varphi^{\pi'} = \exists x.F$ **then**
      win $\leftarrow$ *win*$[x \mapsto \pi_i]$
      conds $\leftarrow$ *conds*$[x \mapsto \pi_i]$
    else if $\varphi^{\pi'} = \forall x.F$ **then**
      win $\leftarrow$ *win*$[x \mapsto H[\pi'\bullet]$
      conds $\leftarrow$ *conds*$[x \mapsto H[\pi'\bullet]$
  lose $\leftarrow$ *lose* $\wedge$ ($\neg$*win*) $\wedge$ *conds*
if *lose is satisfiable* **then**
  Let $M \models$ *lose*$\{x \mapsto M_0(x) : x \in dom(M_0)\}$
  $\langle U, G, wins\rangle \leftarrow$ **CSS**($\varphi, M, M_0, \epsilon, S$)
  return *Counter-strategy* U
return *None*



Since the losing formula is satisfiable, there must be some counter-strategy that beats S. One such counter-strategy U is depicted to the right—remember that the UNSAT strategy U is a SAT strategy to the formula $\neg\varphi$. As in Ex. 1, U is annotated with additional labels: terms are labeled with the existential quantifier they are instantiating, each $\bullet$ is annotated with the Herbrandized universal quantifier they represent, and conjunctions are grouped and highlighted to visually distinguish conjunctive branches from disjunctive branches. Finally, each leaf of the skeleton is labeled with the atomic formula reached after substituting the terms and Herbrand constants chosen for the existential and universal quantifiers, respectively.

The skeleton U states that UNSAT will always choose 0 to instantiate x and 1 to instantiate z. If SAT chooses the left branch, then the play is over and UNSAT wins. Otherwise, SAT chooses the right branch and a symbolic value $\bar{y}$ to instantiate y. Then SAT chooses to either play the left or right branch of the resulting sub-game. If SAT chose left then UNSAT will chose to play the left sub-game and the play ends in the atom $0 \geq \bar{y}$. Otherwise, when SAT plays the right sub-game, UNSAT chooses to play the resulting left sub-game and play ends in the atom $1 \geq \bar{y}$.

Algorithm 2 constructs M form the losing formula of S for the game $G(\varphi, M_0)$, then uses Algorithm 3 to construct U from S, φ, and the model M.

Algorithm 3: Constructing a counter-strategy

Function CSS(φ, M, M^π, π, S)

Input: LRA formula $\varphi = \psi^\pi$ for some ψ.
Valuation $M : \mathit{Image}(H) \to (\mathbb{Q} \cup \mathbb{B})$
Valuation $M^\pi : (x_0, \ldots, x_n) \to \mathbb{Q}$ s.t.
$FV(\varphi) \subseteq \mathit{dom}(M^\pi)$
π a path fixing SAT's initial moves
S the strategy skeleton for φ
Output: $\langle U, G \rangle$
if $M \models \mathit{lose}(\varphi, S)$ then
U is an unsat skeleton that beats S from any state satisfying $M' \models G$, where $M^\pi \models G$.
if $S = \emptyset$ **then**
  return
  $\langle$Any $\mathit{skel} \in \mathit{SKEL}(\neg\varphi^\pi, \mathit{dom}(M^\pi)), \top\rangle$
else if φ^π *is atomic* **then**
  return $\langle\{\epsilon\}, \neg\varphi^\pi\rangle$
else if $\varphi^\pi = F \wedge G$ **then**
  if $\neg[\![H[\pi \cdot L]]\!]^M$ **then**
    $\langle U_l, G_l\rangle \leftarrow$ **CSS**$(F, M, M^\pi, \pi \cdot L, \{\pi : L \cdot \pi \in S\})$
    return $\langle\{L \cdot \pi : \pi \in U_l\}, G_l\rangle$
  else
    $\langle U_r, G_r\rangle \leftarrow$ **CSS**$(G, M, M^\pi, \pi \cdot R, \{\pi : R \cdot \pi \in S\})$
    return $\langle\{R \cdot \pi : \pi \in U_r\}, G_r\rangle$
else if $\varphi^\pi = F \vee G$ **then**
  $\langle U_l, G_l\rangle \leftarrow$ **CSS**$(F, M, M^\pi, \pi \cdot L, \{\pi : L \cdot \pi \in S\})$
  $\langle U_r, G_r\rangle \leftarrow$ **CSS**$(G, M, M^\pi, \pi \cdot R, \{\pi : R \cdot \pi \in S\})$
  return
  $\langle\{L \cdot \pi : \pi \in U_l\} \cup \{R \cdot \pi : \pi \in U_r\}, G_l \wedge G_r\rangle$
else if $\varphi^\pi = \forall x.F$ **then**
  $M^{\pi\bullet} \leftarrow M^\pi\{x \mapsto [\![H[\pi\bullet]]\!]^M\}$
  $\langle U, G\rangle \leftarrow$ **CSS**$(F, M, M^{\pi\bullet}, \pi\bullet, \{\pi : \bullet\pi \in S\})$
  $t \leftarrow$ **select**$(M^{\pi\bullet}, x, G)$
  return $\langle\{t \cdot \pi : \pi \in U\}, G[x \mapsto t]\rangle$
else if $\varphi^\pi = \exists x.F$ **then**
  $U \leftarrow \emptyset$
  $G \leftarrow \mathit{true}$
  for $S \to^t S'$ **do**
    $M^{\pi \cdot t} \leftarrow M^\pi\{x \mapsto [\![t]\!]^{M^\pi}\}$
    $\langle U^+, G^+\rangle \leftarrow$ **CSS**$(F, M, M^{\pi \cdot t}, \pi \cdot t, S)$
    $G \leftarrow G \wedge (G^+[x \mapsto t])$
    $U \leftarrow U \cup U+$
  return $\langle\{\bullet\pi : \pi \in U\}, G\rangle$

The algorithm first constructs the losing formula. The first step of which introduces a new Herbrand constant for each path to a universal quantifier and a fresh Boolean variable for each path to a conjunct within φ. The produced formula is equisatisfiable to the losing formula described in Section 3. By existentially quantifying the introduced boolean variables *lose* becomes logically equivalent to $\mathit{lose}(S, \varphi)$. The introduced boolean variables enables an explicit encoding UNSAT's choice of conjunct as a variable a model can choose between. After constructing *lose*, Algorithm 2 checks if the formula is satisfiable. If *lose* is unsatisfiable, then S is a winning skeleton for the (sub-)game φ and has no counter-strategy. Otherwise, there is a model of *lose*, that can be used with Algorithm 3 to produce an UNSAT skeleton that beats S.

Algorithm 3 recursively decomposes S and φ to produce a counter-strategy. Before recursing, a model of the bound variables (M^π) and the path-prefix π is constructed. For universals, the valuation is extended using the model of *lose*, and for existentials the valuation is extended by evaluating the term instantiating the quantifier using the model of the previously bound variables. To ensure that the recursive call produces a counter-skeleton that beats the sub-skeleton of S, M must be a model of the losing formula of the sub-skeleton. Whenever φ is not conjuntive this is trivially true as $\mathit{lose}(\varphi, S)$ is a conjunction of the losing formulae for the sub-skeletons. This ensures that the model of the parent formula is also a model of all sub-formulae. In the case when φ is a conjunction, the

boolean variables introduced to construct the losing formula determine which of the subformulae are also modeled by M. If the introduced boolean variable for a given conjunct is false in the model M, then it must be that the given conjunct also evaluates to false in the given model. Thus is is ensured that whenever the recursive call is made to Algorithm 3, the returned counter-strategy is indeed a counter-strategy to the sub-skeleton from any state that models G. The algorithm then goes back up and constructs a counter-strategy. For atomic formula, there is only one possible strategy, the empty strategy. For conjuncts, the counter-strategy simply extends a counter-strategy for the left or right branch of the conjunct depending on which branch has a counter-strategy in model M—it is possible both have a counter strategy in model M, taking either or both counter-strategies produces a counter-strategy. For disjunctions, a counter-strategy combines a counter-strategy for both the left and right disjuncts. If the strategy S only takes one of the two branches, then any skeleton for the disjunct may be returned. For universal quantifiers, we use model based term selection to select a term t to instantiate x such that t satisfies the same atoms of G within the given module $M^{\pi\bullet}$. For existentials, we construct a counter-strategy as the union of a counter-strategy for each choice SAT had made.

6 Synthesizing Fine-grained Strategies

Section 4 presents a fine-grained strategy improvement algorithm that computes a winning strategy skeleton that either proves or refutes satisfiability of a LRA sentence. This section shows how to generalize the technique of [8] to compute a winning fine-grained strategy from a winning fine-grained strategy skeleton. As described in Section 2 a SAT strategy is a function from plays to either a rational number (for existential quantifiers) or the labels L and R (for disjunctions).

We construct a system of constrained horn clauses (CHCs) whose solution we may use to produce a winning strategy from a winning skeleton. The produced CHC rules represent when the strategy skeleton is losing. Since the strategy skeleton is winning, the rules are unsatisfiable and a labeling to prove false for the set of CHCs exists. The process starts by labeling each node—a node of S is any path π such that $\pi\pi' \in S$—of the strategy skeleton. Each non-atomic node is labeled with $\top$ and each atomic node (leaf of the skeleton) is labeled with the atom reached, substituting each of the terms instantiating existential quantifiers. Formally, for leaf of the skeleton whose path from the root is π is labeled with the formula $\mathit{subst}_\varphi(\neg\varphi^\pi, \pi)$, where φ is the LRA-satisfiablity game being played. The function subst_φ applies a substitution based on the given path in reverse order of the appearance of each existential quantifier. As CHCs are locally scoped, we may avoid Herbrandization (i.e. uniquely renaming each

occurrence of a universal variable based on binding location).

$$\begin{aligned}
subst_\varphi(G, \epsilon) &\triangleq G \\
subst_\varphi(G, \pi \cdot \mathrm{in}_*) &\triangleq subst_\varphi(G, \pi) \\
subst_\varphi(G, \pi\bullet) &\triangleq subst_\varphi(G, \pi) \\
subst_\varphi(G, \pi \cdot t) &\triangleq subst_\varphi(G[x \mapsto t], \pi) \qquad \text{where } \varphi^\pi = \exists x.F
\end{aligned}$$

The set of CHCs associate a relation and a set of variables with each node of the skeleton. The set of variables associated with each node π is the free variables of φ^π. Let R_n denote the relation for node n in the strategy skeleton. We produce the following rules:

$$\begin{aligned}
&subst_\varphi(\neg\varphi^\pi, \pi) \Rightarrow R_n(\dots) && \text{if } n \text{ is an atomic node at path } \pi \\
&\left(\bigvee_{R_{n'}} R_{n'}(\dots)\right) \Rightarrow R_n(\dots) && \text{if } n \text{ is conjunctive} \\
&\left(\bigwedge_{R_{n'}} R_{n'}(\dots)\right) \Rightarrow R_n(\dots) && \text{otherwise}
\end{aligned}$$

Each R_n represents the condition under which the skeleton rooted at node n loses. Since the overall skeleton is winning, the rules are unsatisfiable and have a solution. A solution for each relation R_n may be computed using an off-the-shelf CHC solver. The formula produced as the solution for R_n, is the condition under which the skeleton rooted at n is losing. Applying the negated solution as a gaurd for each path of the skeleton, produces a winning strategy. Technically, the guards should be determinized to produce a function; however, any such determinization will result in a winning strategy. Formally, for each path π we produce the function:

$$f_\pi(x_1, ..., x_k) \text{ if } \neg R_{n_1} \text{ then } l_1 \text{ elif } \dots \text{ else } l_{n_m}$$

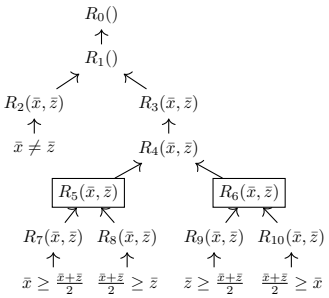
where n is the node rooted at path π within the skeleton, has m children where child n_i is reached with label l_i, and is in scope of the bound variables $x_1, \dots, x_k$.

Consider the winning skeleton S_2 from Example 3. The left side of Example 5 shows the set of rules to label S_2, depicted as a tree (whose shape follows exactly from the shape of S_2). The graph should be interpreted as saying that a node's label is implied by the combination of each of its children's labels. For nodes 5 and 6, the labels of its children should be combined using disjunctive, otherwise, the label of its children should be combined conjunctively. For example, the rule for node 1, should be read as $R_2(\bar{x}, \bar{z}) \wedge R_3(\bar{x}, \bar{z}) \Rightarrow R_1()$, while the rule for node 5 should be read as $R_7(\bar{x}, \bar{z}) \vee R_8(\bar{x}, \bar{z}) \Rightarrow R_5(\bar{x}, \bar{z})$. The middle column of Example 5 shows a possible solution to the set of rules, and the left-hand

side shows the winning strategy extracted from S_2 using the given solution. The strategy $f_{\bullet\bullet}$ states that given UNSAT's choices of $\bar{x}$ and $\bar{z}$ to instantiate x and z, if UNSAT chose equal values for x and y then SAT will chose the left branch—which results in SAT's immediate win—otherwise SAT will chose to play the right branch. $f_{\bullet\bullet R}$ and $f_{\bullet\bullet R\frac{\bar{x}+\bar{z}}{2}}$ are interpreted similarly.

Example 5.

Rules:



Labels:

$R_0() \mapsto \bot$
$R_1() \mapsto \bot$
$R_2(\bar{x},\bar{z}) \mapsto \bar{x} \neq \bar{z}$
$R_3(\bar{x},\bar{z}) \mapsto \bar{x} = \bar{z}$
$R_4(\bar{x},\bar{z}) \mapsto \bar{x} = \bar{z}$
$R_5(\bar{x},\bar{z}) \mapsto \bar{x} \geq \frac{\bar{x}+\bar{z}}{2} \vee \frac{\bar{x}+\bar{z}}{2} \geq \bar{z}$
$R_6(\bar{x},\bar{z}) \mapsto \bar{z} \geq \frac{\bar{x}+\bar{z}}{2} \vee \frac{\bar{x}+\bar{z}}{2} \geq \bar{x}$
$R_7(\bar{x},\bar{z}) \mapsto \bar{x} \geq \frac{\bar{x}+\bar{z}}{2}$
$R_8(\bar{x},\bar{z}) \mapsto \frac{\bar{x}+\bar{z}}{2} \geq \bar{z}$
$R_9(\bar{x},\bar{z}) \mapsto \bar{z} \geq \frac{\bar{x}+\bar{z}}{2}$
$R_{10}(\bar{x},\bar{z}) \mapsto \frac{\bar{x}+\bar{z}}{2} \geq \bar{x}$

Strategy:

$f_{\bullet\bullet}(\bar{x},\bar{z}) \triangleq$ if $\bar{x} = \bar{z}$ then L else R
$f_{\bullet\bullet R}(\bar{x},\bar{z}) \triangleq \dfrac{\bar{x}+\bar{z}}{2}$
$f_{\bullet\bullet R\frac{\bar{x}+\bar{z}}{2}}(\bar{x},\bar{z}) \triangleq$ if $\bar{x} < \bar{z}$ then L else R

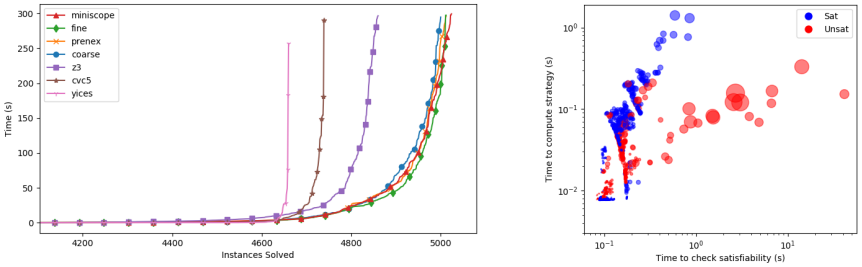
7 Experimental Evaluation

Our experiments aim to answer the following questions: (1) is fine-grained SimSat competitive with state-of-the-art SMT solvers? (2) how much of the difference between coarse-grained SimSat and fine-grained SimSat is driven by considering non-prenex normal form formulas and how much is due to the new strategy improvement algorithm? (3) how much is required to compute a winning fine-grained strategy after checking satisfiability of a formula?

We extend the tool SimSat—a prototype implementation of the coarse-grained strategy improvement algorithm from Farzan and Kincaid [9]—with the fine-grained strategy improvement procedure. SimSat is implemented in OCaml using Z3 [7] to handle ground formulas. We compare fine-grained SimSat to coarse-grained SimSat as well as to Z3 (version 4.11.2) [7], CVC5 (version 1.0.0) [1], and YicesQS [11]. Z3 implements the procedure from Bjørner and Janota [4], CVC5 implements the procedure from Reynolds et al. [16], and YicesQS implements the procedure from Bonacina et al. [5].

We evaluate each tool on three suites of benchmarks: SMT-LIB2, Termination, and Simulation. Each benchmark is described in detail below. All experiments were conducted on a desktop running Ubuntu 18.04 LTS equipped with a 4 core Intel(R) Xeon(R) processor at 3.2GHz and 12 GB of memory. Each experiment was allotted a maximum of five minutes to complete.

To answer (1), fine-grained SimSat is compared to coarse-grained SimSat, CVC5, YicesQS, and Z3. This section does not consider other solvers and methods (e.g. quantifier elimination) as Reynolds et al. [16], Bjørner and Janota [4], and Bonacina et al. [5] show that their methods outperform other existing solvers and methods for quantified LIA and LRA formulas. To answer (2) we consider



(a) A cactus plot showing x instances solved within y seconds per solver.

(b) Log-scale plot of strategy synthesis time (y-axis) vs satisfiability time (x-axis).

Fig. 1

three variants of fine-grained SimSat. The first variant, "prenex," first converts the input formula to prenex-normal form before running the decision procedure. The second variant, "miniscope," miniscopes (reduces the scope of quantifiers) the formula before running the decision procedure, and the final variant, "fine," applies the decision procedure without modifying the input formula. Finally, to answer (3) we wish to measure the efficacy of our algorithm for strategy synthesis, but we know of no other algorithm capable of synthesizing strategies for fine-grained games with which to establish a baseline. Instead, we measure the overhead of synthesizing a strategy on top of synthesizing a strategy skeleton.

SMT-LIB2. This suite of benchmarks consists of 2419 LRA and 616 LIA benchmarks. All benchmarks come from SMT-LIB2 [3]. All LIA benchmarks come from industrial problems. The LRA benchmarks consists of 1800 randomly generated formulas in prenex normal form with varying quantifier depth (see Monniaux [14] for detailed descriptions) and 619 industrial benchmarks.

Termination. This suite of benchmarks consists of 200 LIA formulas. The formulas are derived from Zhu and Kincaid's [18] method for proving termination of programs. Each formula encodes a sufficient condition for which a program is terminating—the program terminates if the formula is valid. The suite of benchmark consists of a formula for each program in the "polybench" and "termination" benchmarks from Zhu and Kincaid's evaluation section [18]. For complete details on formula construction see Sections 5 and 6 of Zhu and Kincaid [18].

Simulation. This suite of benchmarks consists of 2060 LIA formulas. The formulas represent when the state of two integer message passing programs are weakly similar for the next n instructions. For complete details see [citation omitted for anonymity].

Results. Table 1 and Figures 1a 1b summarize the results of the experiments. Figure 1a is a cactus plot. Each line represents a solver's performance. Each point (x, y) within the line for a solver represents that x instances were individually solved in under y seconds by the given solver. The closer the line is to the x-axis the better the solver performed. Table 1 breaks down the results a little further by (sub-)suite of benchmarks. Each entry shows the number of instances from the given suite of benchmarks by the given solver. The "Any" column counts the number of instances solved by *any* of the solvers, while the "All" column counts

Benchmarks	Miniscope	Fine	Prenex	Coarse	CVC5	YicesQS	Z3	Any	All	Total
Simulation	**2060**	**2060**	**2060**	2059	2059	1972	**2060**	2060	1972	2060
UltimateAutomizer	316	316	315	315	**345**	82	242	349	60	372
psyco	**189**	**189**	**189**	**189**	**189**	146	**189**	189	146	189
tptp (LIA)	**46**	**46**	**46**	**46**	**46**	42	**46**	46	42	46
Termination	**200**	**200**	**200**	196	195	0	166	200	0	200
All LIA	2811	2811	2810	2805	**2834**	2242	2703	2850	2220	2867
Mjollnir	1597	1584	1586	1578	1300	**1800**	1541	1800	1177	1800
keymaera	**222**	**222**	**222**	**222**	**222**	**222**	**222**	222	222	222
Scholl	372	373	372	373	362	**374**	372	374	359	374
tptp (LRA)	**23**	**23**	**23**	**23**	**23**	**23**	0	23	0	23
All LRA	2214	2202	2203	2196	1907	**2419**	2135	2419	1781	2419
All	**5025**	5013	5013	5001	4741	4661	4861	5269	4001	5286

Table 1: Number of instances solved per suite of benchmarks—UltimateAutomizer, psyco, tptp, Mjollnir, keymaera, and Scholl are subcategories of SMT-LIB2.

the number of instances solved by every solver. The "total" column details the total number of instances (solved and unsolved) within the suite of benchmarks. For each suite of benchmark, bolded values highlight which solver(s) solved the most instances of that set of benchmarks.

Overall, all solvers performed well. In fact, Figure 1a shows that all solvers solved the first 4200 instances in under a second. The Figure zooms into the x-axis after this point to highlight the differences between solvers. The experiments show that the SimSat variants all behaved similarly and out-performed CVC5, Z3, and YicesQS overall. Of the SimSat variants, the miniscoped variant performed best, followed by the normal fine-grained variant, then the fine-grained prenex variant and lastly the coarse-grained variant.

Looking into each suite of benchmarks further, Table 1 shows that while YicesQS solved the fewest instances overall, it actually solved all LRA formulas. Similarly, while CVC5 performed the worst on LRA, it was the best performer on LIA instances, solving 23 more LIA instances than the miniscoped SimSat variant—the next best performer. In both scenarios, the miniscoped SimSat variant placed a close second. CVC5 performed well on the industrial benchmarks; however, struggled with the randomly generated Mjollnir benchmarks, perhaps due to the bottom-up instantiation strategy of its implemented decision procedure [16]. YicesQS excelled at the LRA formulas but failed to solve many of the simulation and termination benchmarks. Overall, Z3 performed well but struggled more with the UltimateAutomizer and Termination benchmarks—benchmarks where conversion to prenex normal form increased quantifier alternations significantly.

Finally, Figure 1b summarizes the cost of computing a winning fine-grained strategy after checking satisfiability of the given formula—i.e. how much time in seconds did it take to compute a winning strategy from a winning strategy skeleton. Figure 1b, plots a point for each formula within the Simulation benchmark. A point has four associated values: (1) its x position represents how much time is required to prove the formula Sat or Unsat (e.g. time to run "Fine" SimSat variant), (2) it's y position represents the amount of time in seconds

required to compute a winning strategy from a winning strategy skeleton, (3) its size visually quantifies the number of AST-nodes within the produced winning strategy, and (4) a node is blue if the formula is won by SAT and red if it is won by UNSAT. The smallest computed strategy consisted of a single node (move), while the largest strategy consisted of 448 nodes. Across all instances, the it took roughly 18.4% extra time to additionally compute a winning strategy over just determining satisfiability of a formula. The maximum time to compute a strategy is 1.4 seconds.

8 Discussion and Related Works

The closest techniques to Algorithm 1 are the QSMA algorithm of Bonacina et al. [5] and the coarse-grained strategy improvement algorithm of Farzan and Kincaid [8]. Fine-grained and course-grained strategy improvement algorithms are similar in that they both use model-based term selection to synthesize counter-strategies to find better and better strategies for each player; however, they differ in a few key ways. Fine-grain strategy synthesis works for formulae that are not in prenex normal form. Additionally, while the coarse-grained strategy improvement iterates between skeletons for the two players computing a counter-strategy to the previous players most recent skeleton, the fine-grained strategy improvement algorithm chooses a sub-game to focus on and solve before returning to the current game. The coarse-grained algorithm iterates over "global" strategies, where the fine-grained algorithm builds up a strategy by recursively solving sub-games. While Algorithm 1 and QSMA share a similar high-level recursive structure and used model-based techniques, the method of solving sub-formulae differ. The QSMA algorithm uses over- and under-approximations to abstract quantified sub-formulae when determining satisfiability of the current formula whereas Algorithm 1 uses winning strategies of sub-games and model-based term selection to synthesize counter-strategies and ultimately yield a winning strategy to the current formula.

Algorithm 1 also shares some similarities with QSAT the quantified satisfiability algorithm of Bjørner and Janota [4] which is also based on the game semantics of FOL. For formulas in prenex normal form, QSAT and Algorithm 1 both fix a strategy for the first quantifier and then recursively compute a strategy for the remaining quantifiers and back-tracks if no winning strategy exists for the current player; however, the notion of strategy used differs. In QSAT, a strategy selects a subset of the literals in the formula—whose free variables belong to the prefix of quantifiers already explored—that constrains the possible strategies of the remaining quantifiers.

Finally, Algorithm 1 shares similarities with the counter-example instantiation method of Reynolds et al. [16]. Both methods work for formula beyond prenex normal form and use model based projection techniques to instantiate quantifiers; however, Algorithm 1 uses a top-down approach to synthesize winning strategies, while counter-example instantiation uses a bottom-up technique to instantiate and eliminate quantifiers one quantifier block at a time.

Other methods for LRA/LIA formulas include heuristic instantiation and quantifier elimination. Heuristic instantiation is sound but incomplete and was traditionally the method of choice for many SMT solvers (e.g. CVC4 [2]). Traditional quantifier elimination methods (e.g. Fourier-Motzkin eliminiation [13], Ferrante-Rackoff [10], and Wespfenning [17] algorithms for LRA, and Cooper's algorithm [6], and Pugh's Omega test [15] for LIA) are sound and complete for LRA/LIA but are extremely costly. Recently, Monniaux developed a lazy quantifier elimination method for LRA based on polyhedral projection that performs better in practice [14]. However, Bjørner and Janota show that their algorithm dominates Monniaux's methods [4].

References

1. Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., et al.: cvc5: a versatile and industrial-strength smt solver. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 415–442. Springer (2022)
2. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: Cvc4. In: International Conference on Computer Aided Verification. pp. 171–177. Springer (2011)
3. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org` (2016)
4. Bjørner, N.S., Janota, M.: Playing with quantified satisfaction. LPAR (short papers) **35**, 15–27 (2015)
5. Bonacina, M.P., Graham-Lengrand, S., Vauthier, C.: Qsma: a new algorithm for quantified satisfiability modulo theory and assignment. In: International Conference on Automated Deduction. pp. 78–95. Springer (2023)
6. Cooper, D.C.: Theorem proving in arithmetic without multiplication. Machine intelligence **7**(91-99), 300 (1972)
7. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
8. Farzan, A., Kincaid, Z.: Linear arithmetic satisfiability via strategy improvement. In: IJCAI. pp. 735–743 (2016)
9. Farzan, A., Kincaid, Z.: Strategy synthesis for linear arithmetic games. Proceedings of the ACM on Programming Languages **2**(POPL), 1–30 (2017)
10. Ferrante, J., Rackoff, C.: A decision procedure for the first order theory of real addition with order. SIAM Journal on Computing **4**(1), 69–76 (1975)
11. Graham-Lengrand, S.: Yices-qs 2022, an extension of yices for quantified satisfiability (2022)
12. Hintikka, J.: Game-theoretical semantics: insights and prospects (1982)
13. Kroening, D., Strichman, O.: Decision procedures. Springer (2016)
14. Monniaux, D.: Quantifier elimination by lazy model enumeration. In: International Conference on Computer Aided Verification. pp. 585–599. Springer (2010)
15. Pugh, W.: The omega test: a fast and practical integer programming algorithm for dependence analysis. In: Supercomputing'91: Proceedings of the 1991 ACM/IEEE conference on Supercomputing. pp. 4–13. IEEE (1991)

16. Reynolds, A., King, T., Kuncak, V.: Solving quantified linear arithmetic by counterexample-guided instantiation. Formal Methods in System Design **51**(3), 500–532 (2017)
17. Weispfenning, V.: The complexity of linear problems in fields. Journal of symbolic computation **5**(1-2), 3–27 (1988)
18. Zhu, S., Kincaid, Z.: Termination analysis without the tears. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. pp. 1296–1311 (2021)

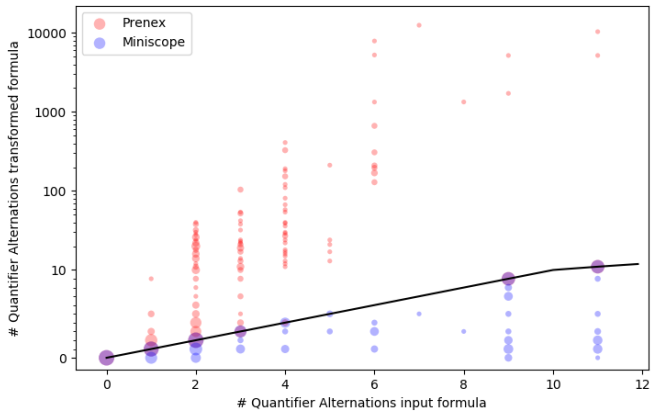


Fig. 2: A plot relating the number of quantifier alternations in the original formula (x-axis) to the number of quantifier alternations in the transformed formulas (y-axis). Blue points represent miniscoping, and red points conversion to prenex normal form. The size of each point's radius scales logarithmically with the frequency of the point (x, y).

A Implementation Details

In Algorithm 3, when countering a conjunction and both branches have a counter-strategy in the given model, we arbitrarily chose the left branch. While it is fine to choose either or both, Algorithm 1 assumes only one branch is chosen. The implementation follows the choice made in Algorithm 3; however, other options may incorporate some heuristic to choose between the branches when both can produce a counter-strategy (e.g. choose the more "constrained" branch or the branch with fewer quantifier alternations, etc.). The term selection function *select* in Algorithm 3, is the same as the one used in coarse SimSat—as described in Farzan and Kincaid [8]. When Algorithm 1 and 3 chose any skeleton for a formula, the implementation initializes a formula using the hueristic described in Section 4. Specifically, the implementation will first initialize an UNSAT skeleton for φ that instantiates all universals quantifiers with 0 and always chooses the left branch of any conjunction. Then the implementation will find a counter-strategy to the default initialized UNSAT skeleton.

B Extended Results Discussion

To better understand why the SimSat variants all performed similarly, we performed a deeper analysis of the results, specifically looking into the "shape" of the input formulas. In all but 640 of the instances, the input formula was presented in prenex normal form. Of the 640 instances not in prenex normal form, the conversion to prenex normal form introduces a median of 2 extra quantifier alternations on average. For formulas not in prenex normal form, the number of

alternations presented is the maximum number of quantifiers alternation along any path of the formula's AST. Miniscoping reduced the number of quantifier alternations for 718 of the formulas. The median reduction resulted in 1 fewer quantifier alternation. Figure 2 provides a more detailed picture relating the number of quantifier alternations in the transformed formula to the number in the original formula. Each red point (x, y) represents that the input formula had x quantifier alternations, while it's prenex conversion had y quantifier alternations. Blue points similarly relate the input formula to its miniscoped version. Larger points represent a greater number of formulas at point (x, y). The graph shows that most formulas have a point that falls on the line $x = y$, while the remainder of the points typically have a frequence of 1. The figure shows that while rare in these benchmarks, conversion to prenex normal form can yield substantially more quantifier alternations.

C Proofs

Proposition 1. *Let φ be a LRA sentence and S a SAT strategy skeleton for φ. S is winning if and only if lose(S, φ) is unsatisfiable.*

Proof. We first extend the definition of winning to skeletons for formulas with free-variables.

Let φ be an LRA formula, M a valuation over the free variables of φ.

For any play π of φ, π is won by SAT from M if and only if $M^\pi \uplus M \models \varphi^\pi$.

Similarly, a SAT strategy f is winning from M if and only if every complete play conforming to f is won by SAT from M.

Lastly, a SAT skeleton S is winning from M if and only if there is a strategy that conforms to S that is winning from M.

We now proceed to prove a more general theorem:

Let φ be a LRA sentence, M a valuation over the free variables of φ, and S a SAT skeleton for φ. S is winning from M if and only if $M \not\models \text{lose}(S, \varphi)$.

We now proceed to prove the general theorem by induction on φ.

Case (φ is atomic). S must be $\{\epsilon\}$, and lose$(\{\epsilon\}, \varphi) = \neg\varphi$.

Clearly, S is winning from M if and only if $M \not\models \neg\varphi$.

Case ($\varphi \wedge \psi$). By construction $S = (L \cdot S_L) \cup (R \cdot S_R)$ for some S_L and S_R

Necessarily, M is a valuation for the free variables of both φ and ψ.

Case ($\Rightarrow$). By Assumption S is winning from M for $\varphi \wedge \psi$.

Necessarily, S_L is winning from M for φ and similarly for S_R and ψ. By the IH, we may conclude that lose(S_L, φ) and lose(S_R, ψ) are not satisfied by M. By definition, lose$(S, \varphi \wedge \psi) =$ lose$(S_L, \varphi) \vee$ lose(S_R, φ). Thus, it is also unsatisfied by M.

Case ($\Leftarrow$). By Assumption $M \not\models \text{lose}(S, \varphi \wedge \psi)$.

We may conclude that neither lose(S_L, φ) nor lose(S_R, ψ) are satisfied by M. Using the IH, we conclude that S_L is winning from M for φ and

similarly for S_R and ψ. Since both branches of S are winning from m, S must be winning from M.

Case $(\varphi \vee \psi)$. By construction, $S = (L \cdot S_L) \cup (R \cdot S_R)$ for some S_L and S_R with possibly one of S_L or S_R being empty.

Case $(\Rightarrow)$. By assumption, S is winning from M.

By definition, there is some strategy f that conforms to S that is winning from M. Let $f_L = f(\pi)$ for every path $(L \cdot \pi) \in \mathit{dom}(f)$. f_R is similarly defined. Either f_L is empty or f_L is winning from M for φ. In the latter case, by definition S_L is winning form M and we may use the IH, to conclude that M is not a model of $\mathrm{lose}(S_L, \varphi)$, which necessarily implies that M is not a model of $\mathrm{lose}(S, \varphi \vee \psi)$. In the case where $\mathit{dom}(f_L)$ is empty, then $\mathit{dom}(f_R)$ is non-empty; otherwise, $\mathit{dom}(f)$ is empty and thus not a winning strategy. By similarly reasoning f_R is winning from M, and we may conclude that M is not a model of $\mathrm{lose}(S, \varphi \vee \psi)$.

Case $(\Leftarrow)$. By assumption, M is not a model of $\mathrm{lose}(S, \varphi \vee \psi)$.

By definition, $\mathrm{lose}(S, \varphi \vee \psi) = \mathrm{lose}(S_L, \varphi) \wedge \mathrm{lose}(S_R, \psi)$. Thus, either M is not a model of both $\mathrm{lose}(S_L, \varphi)$ and $\mathrm{lose}(S_R, \psi)$. In either case, we may apply the IH to conclude that either S_L is winning from M or S_R is winning from M. Since at least one branch must be winning from M, S must be winning from M as well.

Case $(\forall x.\ \varphi)$. By construction $S = \bullet \cdot S'$ and $\mathrm{lose}(S, \forall x.\ \varphi) = \exists x.\ \mathrm{lose}(S', \varphi)$.

Case $(\Rightarrow)$. By assumption, S is winning from M.

Proof by contradiction. Suppose M is a model of $\mathrm{lose}(S, \forall x.\ \varphi)$. There must be some value c such that $M\{x \mapsto c\}$ is a model of $\mathrm{lose}(S', \varphi)$. By the IH, it must be that S' is not winning from $M\{x \mapsto c\}$. However, by assumption S is winning from M, and thus S' must be winning from $M\{x \mapsto c\}$, a condradiction.

Case $(\Leftarrow)$. By assumption, $M \not\models \mathrm{lose}(S, \forall x.\ \varphi)$.

Thus for any possible value c, $M\{x \mapsto c\}$ is not a model of $\mathrm{lose}(S', \varphi)$. By the IH, we may conclude that S' is winning from $M\{x \mapsto c\}$. Since c is arbitrary, it must be that S is winning from M.

Case $(\exists x.\ \varphi)$. By construction $S = (t_0 \cdot S_0) \cup \cdots \cup (t_n \cdot S_n)$ and $\mathrm{lose}(S, \exists x.\ \varphi) = \mathrm{lose}(S_0, \varphi)[x \mapsto t] \wedge \cdots \wedge \mathrm{lose}(S_n, \varphi)[x \mapsto t_n]$.

Case $(\Rightarrow)$. By assumption S is winning from M.

By definition there is some winning strategy f that conforms to S. $f(\epsilon)$ must be some constant c. Let $f'(\pi) = f(c \cdot \pi))$ for every $c \cdot \pi \in \mathit{dom}(f)$. Necessarily, there must be some i sucht that $[\![t_i]\!]_M = c$ and f' conforms to S_i. f' is winning from $M\{x \mapsto f(\epsilon)\}$. Thus, S_i is winning from $M\{x \mapsto f(\epsilon)\}$. By the IH, $\mathrm{lose}(S_i, \varphi)$ is not modeled by $M\{x \mapsto c\}$. Since $[\![t_i]\!]_M = c$, we may conclude that M is not a model of $\mathrm{lose}(S_i, \varphi)[x \mapsto t_i]$. Finally, we may conclude that M is not a model of $\mathrm{lose}(S, \varphi)$.

Case $(\Leftarrow)$. By assumption M is not a model of $\mathrm{lose}(S, \exists x.\ \varphi)$.

There must be some S_i and t_i such that $M \not\models \mathrm{lose}(S_i, \varphi)[x \mapsto t_i]$. Thus $M\{x \mapsto [\![t]\!]_M\} \not\models \mathrm{lose}(S_i, \varphi)$. By the IH, S_i must be winning from $M\{x \mapsto [\![t]\!]_M\}$. There must be some strategy f' that conforms to S_i that wins

from $M\{x \mapsto [\![t]\!]_M\}$. Let $f(\epsilon) = [\![t]\!]_M)$ and $f([\![t]\!]_M \cdot \pi) = f'(\pi)$ for every $\pi \in dom(f')$. f is winning from M. f conforms to S. Thus, S is winning from M.

□

Theorem 1. *Algorithm 1 is a decision procedure for LRA satisfiability.*

Proof. We proceed by proving a more general theorem:
Let φ be an LRA formula whose top level connective is a universal quantifier or a conjunction, M be a model of the free variables of φ ($FV(\varphi) \subseteq dom(M)$), and S be a SAT skeleton for φ. (1) **Solve**(φ, M, S) is terminating, (2) if **Solve**(φ, M, S) returns *Sat* S', then S' is a SAT skeleton that is winning from M, and (3) if **Solve**(φ, M, S) returns *Unsat* U, then U is an UNSAT skeleton that is winning from M.

Necessarily, Algorithm 1 is a decision procedure if the more general theorem holds. We proceed to prove the more general theorem by induction on the size of φ.

Case (φ is atomic). S must be $\{\epsilon\}$.

Either S has a counter-strategy from M or it does not. In the second case, **Solve** returns *Sat* S. It must have been the case that $M \not\models \text{lose}(S, \varphi)$. We may then use Proposition 1, to conclude that S is winning from M.

In the first case, S must have a counter-strategy. Let U be the returned counter-strategy. By definition, U must be $\{\epsilon\}$, π' is ϵ and U' is U. **Solve** recursively calls **Solve**$(\neg\varphi, M, U)$. From the fact that S had a counter-strategy we may conclude that $M \models \text{lose}(\varphi, S)$. By definition, $\text{lose}(\varphi, S) = \varphi$. It must be that $M \not\models \text{lose}(\neg\varphi, U)$ as $\text{lose}(\neg\varphi, U) = \neg\varphi$. U does not have a counter-strategy. Thus the recursive call will return *Sat* U, and the current call to **Solve** willl return *Unsat* U. Necessarily, U is a winning UNSAT skeleton from M.

Case (φ is not atomic).

Either S has a counter-strategy from M or it does not. In the second case, **Solve** returns *Sat* S. Since S did not have a counter-strategy $\text{lose}(S, \varphi)$ must not be modeled by M. Thus S must be a winning SAT skeleton from M.

Otherwise, S has a counter-skeleton U. By the assumption that φ must begin with either a universal quantifier or a conjunction, π' must be non-empty, and $U' = \{\pi : \pi'\pi \in U\}$. **Solve** recursively calls **Solve**$(\neg\varphi^{\pi'}, M \cup M^{\pi'}, U')$. By the IH, the recursive call is terminating and either returns a winning SAT skeleton from $M \cup M^{\pi'}$ or a wining UNSAT skeleton from $M \cup M^{\pi'}$. If the recursive call returns a SAT skeleton U'', then the current call to **Solve** returns *Unsat* $\{\pi'\pi : \pi \in U''\}$. Since U'' is a winning SAT skeleton for $\neg\varphi^{\pi'}$ from $M \cup M^{\pi'}$, U'' must be a winning UNSAT skeleton for $\varphi^{\pi'}$. Since $M^{\pi'}$ exactly records the choices of π', it must be the case that the returned UNSAT skeleton from M.

If the recursive call returns an UNSAT skeleton S' from $M \cup M^{\pi'}$, then the algorithm does not yet terminate, and instead returns the result of calling **Solve**$(\varphi, M, S \cup \{(\neg\pi')\pi : \pi \in S'\})$. From the above reasoning, it's clear that

if the new call terminates, then the returned result will satisfy properties (2) and (3).

We now turn to proving that the algorithm will eventually terminate. Since S' is a winning skeleton for the sub-game $\neg\varphi^{\pi'}$, it must be a counter-strategy to U'. Moreover, the new SAT skeleton passed to the new call to **solve**, $S \cup \{(\neg\pi') \cdot \pi : \pi \in S'\}$, must also beat U. Every path the conforms to both U and the new SAT skeleton must be won by SAT; otherwise, S' would not be winning. By accumulating, the new skeletons on each recursive call, we can be sure that U is never produced as a counter strategy to any future call of **solve**.

More explicitly, a counter-strategy for the current game is never produced more than once—**Solve** always makes progress. There are finitely many counter-strategies producable by Algorithm 3. There are only 2 possible choices for every disjunction (take the left branch or take the right branch). For existentials, there are finitely many possible choices of terms selected by the term selection algorithm. This result is proven by Farzan and Kincaid [8]'s Lemma 4.4. Combining these two facts—no counter-strategy is explored twice and there are finitely many counter-strategies—it must be that **Solve** will eventually terminate.

□