Quantified Linear Arithmetic Satisfiability via Fine-Grained Strategy Improvement

Charlie Murphy[1](✉) and Zachary Kincaid[2]

[1] University of Wisconsin-Madison, Madison,
WI 53706, USA
tcmurphy4@wisc.edu

[2] Princeton University, Princeton, NJ 08540, USA
zkincaid@cs.princeton.edu

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 such that the given formula evaluates 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 and find it performs favorably compared with state-of-the-art solvers.

Keywords: Quantified Satisfiability · SMT · Game Semantics · Strategy Improvement

1 Introduction

Checking satisfiability of quantified formulae modulo 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,

A. Gurfinkel and V. Ganesh (Eds.): CAV 2024, LNCS 14681, pp. 89–109, 2024.
https://doi.org/10.1007/978-3-031-65627-9_5

SMT solvers either perform quantifier elimination, which is often computationally expensive, or heuristically instantiate quantifiers, which is sound but incomplete for deciding satisfiability [19]. Recently, decision procedures have been developed to check satisfiability of quantified LA formulae directly [4,5,8,18]. Notably, both Bjørner and Janota [4]'s and Farzan and Kincaid [8]'s decision procedures are based on the game semantics of first-order logic.

The game semantics of first-order logic gives meaning to a formula as a two player game [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—a function determining how SAT should instantiate existential quantifiers and choose sub-formulae of disjuncts to prove the formula satisfiable—to the induced game.

The game-theoretic view of formulae suggests a variation of the satisfiability problem, in which the goal is not (just) to check satisfiability of a formula, but to synthesize a winning strategy for one of the two players. Strategy synthesis can be used as a decision procedure, but can also used for other tasks where a simple yes or no is insufficient (e.g., program synthesis, angelic symbolic execution, or invariant generation).

While the game semantics of first-order logic gives meaning to both quantifiers and connectives, both Bjørner and Janota [4]'s and Farzan and Kincaid [8]'s decision procedures only make use of the game semantics of quantifiers. To do so, both techniques require the input formula to be in prenex normal form—the formula is a sequence of quantifiers followed by a quantifier free formula. While any formula may be converted into a prenex normal form, doing so 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 *game semantics* of the formula. Since prenex conversion does not preserve game semantics, it cannot be used in applications that rely on *strategy synthesis* rather than a yes/no answer.

Existing techniques for checking satisfiability of LA formulas are incapable of producing strategies for both quantifiers and Boolean connectives [4,5,8,18]. While both Bjørner and Janota [4]'s and Farzan and Kincaid [8,9] use the game semantics of LA formulas, they limit their scope to quantifiers via conversion to prenex normal form. Furthermore, the procedure by Bjørner and Janota does not produce an explicit term used to instantiate quantifiers. On the other hand, while the techniques of both Reynolds et al. [18] and Bonacina et al. [5] exploit the fine-grained (quantifier and Boolean connectives) structure of formulas, they do not produce a winning strategy.

This paper presents a decision procedure for checking satisfiability of quantified LA formulae that exploits the fine-grained structure of a formula to produce a winning strategy for SAT or UNSAT for both quantifiers and Boolean connectives. Our technique *Fine-grained Strategy Improvement* uses the fine-grained structure of LA formulas to formulate a recursive procedure that iteratively improves a candidate strategy via computing winning strategies to induced subgames. 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) allowing extraction of a proof object (a winning strategy) that determines exactly how the formula is proven to be (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 any theory that admits an appropriate term-selection function (cf. Sect. 5.1) including linear integer arithmetic (LIA). In Sect. 2, we review the game semantics for linear arithmetic [12], and its relation with LRA satisfiability. Sections 3, 4, and 5 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 extended version[1] contains implementation details, proofs, and extended experimental results.

2 Fine-Grained Game Semantics for LRA Satisfiability

This section reviews the syntax (Sect. 2.1) of **Linear Rational Arithmetic** (LRA) and its game semantics (Sect. 2.2).

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:

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

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.

A **valuation**, $M : V \to \mathbb{Q}$, maps 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

[1] The extended version of this paper is available at https://pages.cs.wisc.edu/~tcmurphy4/docs/fine_grained_strategy_synthesis.pdf.

$FV(t) \subseteq dom(M)$—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 first-order 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 $FV(\psi) \subseteq dom(M)$.

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

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:

$$\begin{array}{ccc}
\varphi^\epsilon \triangleq \varphi & (\forall x.\varphi)^{c\cdot\pi} \triangleq \varphi^\pi & (\exists x.\varphi)^{c\cdot\pi} \triangleq \varphi^\pi \\
M^\epsilon \triangleq \emptyset & M^{c\cdot\pi} \triangleq M^\pi\{x \mapsto c\} & M^{c\cdot\pi} \triangleq M^\pi\{x \mapsto c\}
\end{array}$$

$$\begin{array}{cccc}
(\varphi \wedge \psi)^{L\cdot\pi} \triangleq \varphi^\pi & (\varphi \wedge \psi)^{R\cdot\pi} \triangleq \psi^\pi & (\varphi \vee \psi)^{L\cdot\pi} \triangleq \varphi^\pi & (\varphi \vee \psi)^{R\cdot\pi} \triangleq \psi^\pi \\
M^{L\cdot\pi} \triangleq M^\pi & M^{R\cdot\pi} \triangleq M^\pi & M^{L\cdot\pi} \triangleq M^\pi & M^{R\cdot\pi} \triangleq M^\pi
\end{array}$$

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 implicitly 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-sentence, 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.

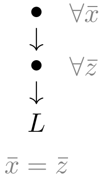
3 Fine-Grained Strategy Skeletons

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 Sect. 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. Sect. 5).

Example 1. To illustrate fine-grained strategy skeletons and the algorithms presented in this paper consider the formula φ which we use as a running example 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 • 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 left branch of the disjunction leading to the atom $x = z$—at the end of the path we display $\bar{x} = \bar{z}$, which is this atom after substituting the placeholder values for UNSAT's choice for the formally bound variables.

As seen in Examples 1 and 4, 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 variables that in-scope in φ. 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. Similarly, we define $\pi \Downarrow S = \{\pi' : \pi \cdot \pi' \in S\}$ to be the set of suffixes of π appearing in S. Formally, a skeleton is a subset of $(\mathbf{Term}(X) \cup \{\bullet, L, R\})^*$ (whose specific form depends on the formula φ). We define *SKEL* as the least solution to the following set of rules:

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

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

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

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) $\varphi^{\pi_0,\dots,\pi_{i-1}} = \exists x.\psi$ for some ψ and $[\![x]\!]^{M^\pi} = [\![\rho_i]\!]^{M^\pi}$, or (2) $\varphi^{\pi_0,\dots,\pi_{i-1}}$ is a disjunctive or conjunctive formula and $\pi_i = \rho_i$, or (3) $\varphi^{\pi_0,\dots,\pi_{i-1}}$ 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 $G(\varphi, M)$ is winning. To determine if S wins the game $G(\varphi, M)$ we check if the *losing* formula $\text{lose}(S, \varphi)$ is not satisfied by M (i.e., S wins $G(\varphi, M)$ if $M \not\models \text{lose}(S, \varphi)$). This formulation results in a formula that is existentially quantified and can be easily Skolemized to a quantifier free formula and checked with an off-the-shelf SMT solver. Furthermore, we show in Sect. 5 that a model of the Skolemized formula can be used to construct an UNSAT strategy skeleton for φ that beats S. We define $\mathrm{lose}(S, \varphi)$ as follows:

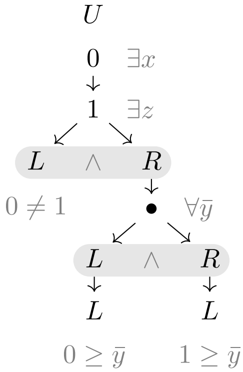
$$\begin{aligned}
\mathrm{lose}(\emptyset, \varphi) &\triangleq \mathit{true} \\
\mathrm{lose}(\{\epsilon\}, \varphi) &\triangleq \neg\varphi \\
\mathrm{lose}(S, \varphi \vee \psi) &\triangleq \mathrm{lose}(L \Downarrow S, \varphi) \wedge \mathrm{lose}(R \Downarrow S, \psi) \\
\mathrm{lose}(S, \varphi \wedge \psi) &\triangleq \mathrm{lose}(L \Downarrow S, \varphi) \vee \mathrm{lose}(R \Downarrow S, \psi) \\
\mathrm{lose}(S, \exists x.\varphi) &\triangleq \bigwedge_{t \cdot \pi \in S} \mathrm{lose}(t \Downarrow S, \varphi)[x \mapsto t] \\
\mathrm{lose}(S, \forall x.\varphi) &\triangleq \exists x.\mathrm{lose}(\bullet \Downarrow S, \varphi)
\end{aligned}$$

If M satisfies the losing formula $\mathrm{lose}(S, \varphi)$, then S is not a winning strategy skeleton for the game $G(\varphi, M)$. Intuitively, this implies that UNSAT can beat SAT if SAT plays according to any strategy conforming to S. We use the intuition to formalize when an UNSAT strategy skeleton U beats the SAT skeleton S.

Definition 2 (Counter Strategy). *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.*

Crucially, it cannot be the case that U beats S and S beats U. This asymmetry is ensures that the strategy improvement algorithm makes progress towards verifying or falsifying the formula φ.

Example 2. 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 $\mathit{lose}(S) \triangleq \bar{x} \neq \bar{z}$ 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 $M = \{\bar{x} \mapsto 0, \bar{z} \mapsto 1\}$. 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 Example 1, U is annotated with additional labels: terms are labeled with the existential quantifier they are instantiating, each $\bullet$ is annotated with the corresponding Skolem constants from $\mathrm{lose}(S, \varphi)$, 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 Skolem constants for each quantified variable.



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 $1 \geq \bar{y}$.

Proposition 1. *Let S be a SAT strategy for the game $G(\varphi, M)$. S is winning if and only if $M \models lose(S, \varphi)$.*

Algorithm 1: Satisfiability modulo LRA

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

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

4 Fine-Grained Strategy Improvement

This section presents an algorithm for deciding LRA satisfiability (Algorithm 1). At a high level, the algorithm produces a winning strategy skeleton via fine-grained strategy improvement. Algorithm 1 iteratively improves the current player (SAT)'s strategy. Each iteration attempts to compute a counter-strategy for the opposing player (UNSAT), fixes the opposing player's initial moves and recursively solves the resulting sub-game. If the opposing player wins the sub-game, then they win the game, and a winning strategy can be constructed using the synthesized initial moves and the winning strategy for the subgame. If the opposing player loses the subgame, the current player's winning strategy for the subgame is used to improve their strategy. The algorithm then proceeds to the next iteration of the current game and repeats until a winning player can be determined.

Algorithm 1 assumes that UNSAT makes the first move in the game $G(\varphi, M)$. 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. Sect. 5). If there is no counter-strategy, then necessarily SAT's current skeleton S must be winning. Otherwise, UNSAT has a counter-strategy U that beats S. The auxiliary function *peel* uses φ and U to compute π'—the leading universal and conjunctive moves—and U'—the remaining skeleton (i.e. $U = \pi' \cdot U'$). 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:

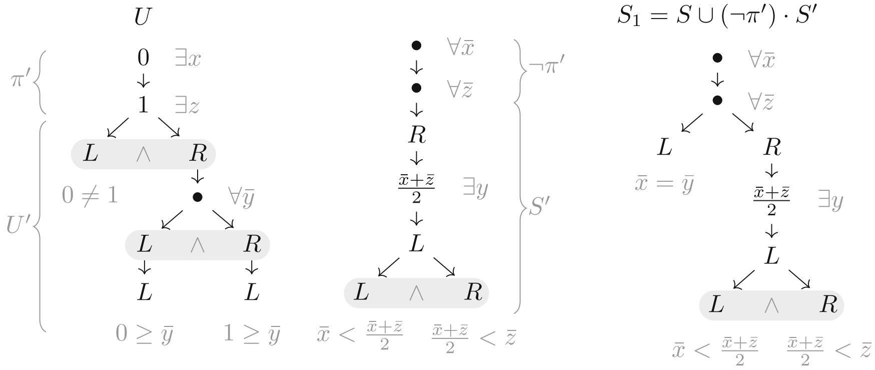
$$\begin{aligned}
\mathit{peel}(\forall x.F, U) &\triangleq \left\langle t \cdot \pi, U' \right\rangle && \text{where } \left\langle \pi, U' \right\rangle = \mathit{peel}(F, U'') \text{ and } U = t \cdot U'' \\
\mathit{peel}(F \wedge G, U) &\triangleq \left\langle L \cdot \pi, U' \right\rangle && \text{where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

By construction, the leading UNSAT moves of a counter-strategy must form a single path—Algorithm 3 only chooses a single term or conjunct when constructing a counter-strategy. This ensures that *peel* is properly defined. After peeling off the leading universal and conjunctive moves from U, the algorithm recursively solves the resulting sub-game (from the point-of-view of UNSAT by recursing on $\neg\varphi^{\pi'}$ instead of $\varphi^{\pi'}$).

After the recursive call, either SAT or UNSAT has a winning skeleton to $G(\neg\varphi^{\pi'}, M^{\pi'})$. If SAT wins $G(\neg\varphi^{\pi'}, M^{\pi'})$ with the skeleton U'', then UNSAT must win $G(\varphi^{\pi'}, M^{\pi'})$ with the UNSAT skeleton U''. Since UNSAT controls the initial moves π', we may conclude that UNSAT wins the entire game $G(\varphi, M)$ and return the winning UNSAT skeleton $\pi' \cdot U''$.

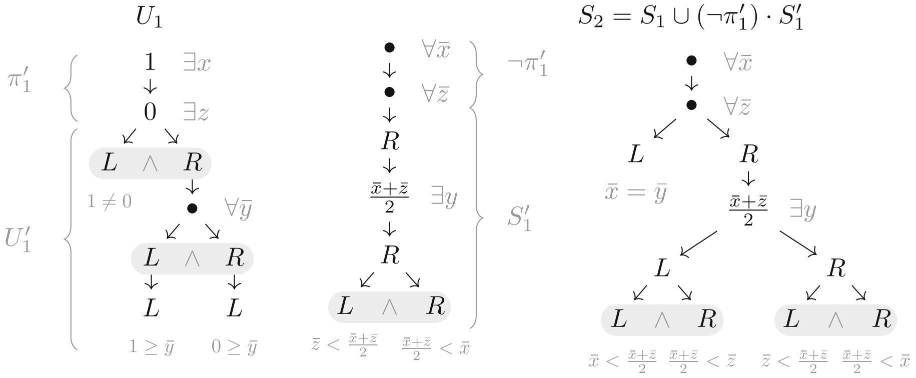
Otherwise, UNSAT wins $G(\neg\varphi^{\pi'}, M^{\pi'})$ with the skeleton S'. The sub-skeleton S' can be extended to counter U by prepending every path of S' with the "negation" of π' the initial moves of UNSAT. Note that by construction, π' consists only of terms instantiating universal quantifiers or L or R denoting a choice of a conjunctive branch. We define the negation of π' as follows: each term in π' is replaced with 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 conjunctive moves—the resulting set of paths only covers one of the branches, while a SAT skeleton must cover both branches of a conjunction—however, when unioned with S the initial skeleton for SAT, the final result is a skeleton that counters U.

Example 3.



Continuing Example 1, let us suppose that we begin Algorithm 1 with the SAT strategy skeleton S depicted in Example 1 (which simply takes the left branch of the disjunction). S is not winning, since the UNSAT player may choose different values for x and z to invalidate the equality—one such counter-strategy U for S appears above. 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.

The recursive call returns that UNSAT won the game $G(\neg\varphi^{\pi'}, M^{\pi\cdot\pi'})$ with the skeleton S'. 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.



While S_1 counters U, it is not yet winning. SAT will lose any play where UNSAT instantiates x and z such that $z < x$. 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

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

```
Function has-counter-strategy($S$, $M_0$, $\varphi$)
    Input: LRA formula $\varphi$.
    Valuation $M_0 : (x_0, \ldots, x_n) \to \mathbb{Q}$ s.t.
    $FV(\varphi) \subseteq dom(M_0)$
    $S$ a SAT strategy skeleton for $\varphi$
    foreach $\pi$ such that $\pi \bullet \pi' \in S$ for some $\pi'$ do
        $H[\pi\bullet] \leftarrow$ fresh rational variable
    foreach $\pi$ such that $\pi L \pi' \in S$ for some $\pi'$ and
    $\varphi^{\pi}$ 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 $M$ be an extension of $M_0$ satisfying $lose$
        $\langle U, G \rangle \leftarrow$ CSS($\varphi, M, M_0, \epsilon, S$)
        return Counter-strategy $U$
    return None
```

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.

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

5 Computing 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.

At a high level, Algorithm 1 uses Algorithm 2 to (1) determine if a strategy skeleton S is winning and (2) if S is not winning to return a counter-strategy U that beats S (and returning none if S is winning). Given a LRA satisfiability game $G(\varphi, M)$ and skeleton S, Algorithm 2 computes (a formula equisatisfiable to) lose(S, φ), then uses Algorithm 3 to synthesize a counter-strategy to S if lose(S, φ) is satisfied by M.

Algorithm 2 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 Sect. 3. By existentially quantifying the introduced Boolean variables and Skolem constants *lose* becomes

Algorithm 3: Constructing a counter-strategy

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

logically equivalent to *lose*(S, φ). The introduced Boolean variables enable an explicit encoding of UNSAT's choice of conjunct within the losing formula. This allows a model of the losing formula (if one exists) to explicitly track which branch UNSAT has a counter-strategy for. The algorithm computes the losing formula on a path-by-path basis. It does so by computing when SAT could win the given path and taking its negation (i.e., $(\neg \mathit{win}) \wedge \mathit{conds}$). We use *win* to denote if the path could be won by SAT—note that for conjunctions SAT must be able to win both of the conjuncts—and *conds* to constrain the introduced Boolean variables (i.e., $H(\pi)$ represents if the sub-skeleton rooted at π is winning). 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 conjunctive this is trivially true, as *lose*(φ, 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. This condition ensures that M is also model of the losing formula of the sub-skeleton and thus that the recursive call computes a counter-strategy to the given sub-skeleton. 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 $M^{\pi\bullet}$ (cf. Sect. 5.1). For existentials, we construct a counter-strategy as the union of a counter-strategy for each choice SAT had made.

5.1 Term Selection

When generating a counter-strategy, Algorithm 3 makes use of the auxiliary function **select** to select a term t to instantiate x. The function **select** is a (model-guided) term selection function [8].

Given a formula F, a variable $x \in FV(F)$ free in F, and a model $M \models F$ of F, we require $\mathbf{select}(M, x, F)$ to return a term t over the free variables of F excluding x (i.e., $FV(t) \subseteq FV(F) \setminus \{x\}$) such that M satisfies F when t is substituted for x (i.e., $M \models F[x \mapsto t]$). Furthermore, to ensure that Algorithm 1 is a decision procedure, we require that for any formula F and variable $x \in FV(F)$ **select** has finite image (i.e., the set $\{\mathbf{select}(M, x, F) : M \models F\}$ is finite).

For LRA, we define **select** as follows. Without loss of generality, we assume that any atom of F that contains x is written as $x = s$, $x < s$, or $x > s$ for some s. Let $EQ(M, x, F)$ contain the term s if and only if $x = s$ is an atom of F and $[\![x]\!]^M = [\![S]\!]^M$. Similarly $UB(M, x, F)$ contains the term s if and only if $x < s$ is an atom of F and $[\![x]\!]^M < [\![s]\!]^M$. Finally, let $LB(M, x, F)$ contain the term s if and only if $x > s$ is an aotm of F and $[\![x]\!]^M > [\![s]\!]^M$. Furthermore, if $EQ(M, x, F)$ is not empty, let $eq(M, x, F)$ be any $s \in EQ(M, x, F)$. If $UB(M, x, F)$ is not empty, then let $lub(M, x, F)$ be a term $s \in UB(M, x, F)$ such that for any other $s' \in UB(M, x, F)$, $[\![s]\!]^M \leq [\![s']\!]^M$. Similarly, if $LB(M, x, f)$ is not empty, then let $glb(M, x, F)$ be a term $s \in LB(M, x, F)$ such that for any other $s' \in LB(M, x, F)$, $[\![s]\!]^M \geq [\![s']\!]^M$.

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

For further details on model-guided term selection (including term selection for linear integer arithmetic), we refer the reader to Farzan and Kincaid [8]. While this paper so far has focused on satisfiability of LRA, we note that Algorithm 1 is a decision procedure for any theory that admits a term selection function with finite image. In fact, the only change required is to swap **select** with an appropriate term selection function for the desired theory.

6 Synthesizing Fine-Grained Strategies

Section 4 presents an 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 Sect. 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).

Strategies vs Skeletons. In Sects. 3, 4, and 5, our techniques and discussion focused on how to compute a *winning strategy skeleton*. While computing a winning strategy skeleton is sufficient to determine satisfiability of a formula, it may be insufficient for other tasks. For example for use in program verification and synthesis tasks (e.g., to determinize non-deterministic choices, synthesize safety conditions, etc.). By definition, a strategy skeleton S is winning if *some* strategy g that conforms to S is winning.

Computing Winning Strategies. This section focuses on how to extract a winning strategy from a winning strategy skeleton S for the game $G(\varphi, M)$. To do so, we construct a system of constrained horn clauses (CHCs) whose solution we 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 satisfiable and a model satisfying the CHCs exists. The process starts by labeling each leaf of S with the atom reached, substituting each of the terms instantiating existential quantifiers. Formally, for any path π of S (from the root to a leaf), we label the leaf (rooted at π) with the formula $subst_\varphi(\neg\varphi^\pi, \pi)$. The function $subst_\varphi$ applies a substitution based on the given path in reverse order of the appearance of each existential quantifier.

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

For a strategy skeleton S, define its nodes $N = \{\pi : \exists \pi'.\ \pi\pi' \in S\}$ to be prefixes of paths in S. Furthermore, define $Succ : N \rightarrow 2^N$ to the set of immediate suffixes. For each node π of S, we introduce an uninterpreted relation $R_\pi(x_1, \ldots, x_n)$ where $x_1, ..., x_n = FV(\varphi^\pi)$ are the free variables of φ^π. We produce the following rules:

$$\begin{aligned} subst_\varphi(\neg\varphi^\pi, \pi) \Rightarrow & R_\pi(\ldots) \text{ if } \varphi^\pi \text{ is atomic} \\ \Big(\bigvee_{\pi' \in Succ(\pi)} R_{\pi'}(\ldots)\Big) \Rightarrow & R_\pi(\ldots) \text{ if } \varphi^\pi \text{ is conjunctive} \\ \Big(\bigwedge_{\pi' \in Succ(\pi)} R_{\pi'}(\ldots)\Big) \Rightarrow & R_\pi(\ldots) \quad \text{otherwise} \\ R_\epsilon(x_1, \ldots, x_n) \Rightarrow & x_1 \neq M(x_1) \vee \cdots \vee x_n \neq M(x_n) \end{aligned}$$

For each $\pi \in N$, R_π represents the set of all M' such that $\pi \Downarrow S$ loses the game $G(\varphi^\pi, M')$. Of note, the last rule requires that R_ϵ does not contain M (i.e., that S must win the game $G(\varphi, M)$). Since the overall skeleton is winning, the rules are satisfiable. A solution for each relation R_π may be computed using an off-the-shelf CHC solver. Applying the negated solution as a guard 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 node π such that φ^π is an existential or disjunctive formula, we produce the function:

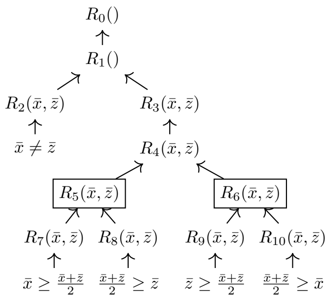
$$f_\pi(x_1, ..., x_k) \text{ if } \neg R_{\pi'_1} \text{ then } l_1 \text{ elif } \ldots \text{ else } l_m$$

where $FV(\varphi^\varphi) \triangleq \{x_1, \ldots, x_k\}$, $Succ(\pi) \triangleq \{\pi'_1, \ldots, \pi'_m\}$, and where each child π'_i is reached with label l_i in S. Furthermore, for each path $\pi \in (\mathbb{Q} \cup \{L, R\})^*$ such that φ^π is an existential or disjunctive formula, define $f'(\pi)$ to be $f_\pi(c_1, \ldots, c_n)$ where each $c_i = M^\pi(x_i)$ for each free variable $x_i \in FV(\varphi^\pi)$. Finally define $f(\pi)$ to be $f'(\pi)$ if $f'(\pi) \in \{L, R\}$ and otherwise define $f(\pi)$ to be $[\![f'(\pi)]\!]^{M^\pi}$. The function f is a strategy conforming to S that wins the game $G(\varphi, M)$.

Consider the winning skeleton S_2 from Example 3. The left side of Example 4 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 4 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 4.

Rules:


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

Labels:

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

Strategy:

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

7 Experimental Evaluation

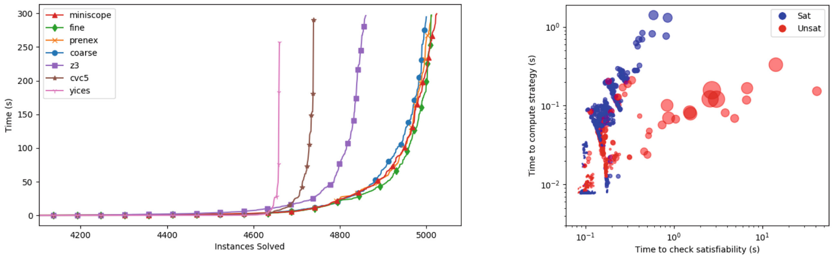
We extend the tool SimSat—a prototype implementation of the coarse-grained strategy improvement algorithm from Farzan and Kincaid [9]—with the fine-grained strategy improvement procedureSimSat is implemented in OCaml using Z3 [7] to handle ground formulas.

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) what is the overhead of computing a winning fine-grained strategy after checking satisfiability of a formula?

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. [18], 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. [18], 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 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,"



(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. (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) (Color figure online)

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 [15] 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 [21] method for proving termination of programs (see Sects. 5 and 6 for details on how formulas are constructed). 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 [21].

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 Chap. 4 of [16], which uses fine-grained strategy synthesis to determinize angelic choice when proving weak simulation between two integer message passing programs.

Results. Table 1 and Figs. 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

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

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

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 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, Fig. 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 [18]. 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, Fig. 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) its 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 s.

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-grained 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 player's 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. [18]. Both methods work for formulas 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 elimination [14], Ferrante-Rackoff [10], and Weispfenning [20] algorithms for LRA, and Cooper's algorithm [6], and Pugh's Omega test [17] for LIA) are sound and complete for LRA/LIA but are extremely costly. Monniaux developed a lazy quantifier elimination method for LRA based on polyhedral projection that performs better in practice [15]. However, Bjørner and Janota show that their algorithm dominates the use of Monniaux's method as a [4]. Finally, Komuravelli et al. [13] introduced model-based projection which under-approximates quantifier elimination for LA and is closely related to the model-based term selection function we use in Sect. 5.1.

Acknowledgements. This work was supported in part by the NSF under grant number 1942537. Opinions, findings, conclusions, or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

References

1. Barbosa, H., et al.: cvc5: a versatile and industrial-strength SMT solver. In: TACAS 2022. LNCS, vol. 13243, pp. 415–442. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_24
2. Barrett, C., et al.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_14
3. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB) (2016). www.SMT-LIB.org
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). https://doi.org/10.1007/978-3-031-38499-8_5
6. Cooper, D.C.: Theorem proving in arithmetic without multiplication. Mach. Intell. **7**(91–99), 300 (1972)
7. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
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. In: Proceedings of the ACM on Programming Languages 2(POPL), pp. 1–30 (2017)

10. Ferrante, J., Rackoff, C.: A decision procedure for the first order theory of real addition with order. SIAM J. Comput. **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. Komuravelli, A., Gurfinkel, A., Chaki, S.: Smt-based model checking for recursive programs. Formal Methods Syst. Des. **48**, 175–205 (2016)
14. Kroening, D., Strichman, O.: Decision procedures. Springer (2016)
15. Monniaux, D.: Quantifier elimination by lazy model enumeration. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 585–599. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_51
16. Murphy, T.C.: Relational Verification of Distributed Systems Via Weak Simulations. Ph.D. thesis, Princeton University (2023)
17. 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)
18. Reynolds, A., King, T., Kuncak, V.: Solving quantified linear arithmetic by counterexample-guided instantiation. Formal Methods Syst. Des. **51**(3), 500–532 (2017)
19. Reynolds, A., Tinelli, C., Goel, A., Krstić, S., Deters, M., Barrett, C.: Quantifier instantiation techniques for finite model finding in SMT. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 377–391. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38574-2_26
20. Weispfenning, V.: The complexity of linear problems in fields. J. Symb. Comput. **5**(1–2), 3–27 (1988)
21. 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)