Improving Strategies via SMT Solving

Thomas Martin Gawlitza, David Monniaux

Improving Strategies via SMT Solving *

Thomas Martin Gawlitza David Monniaux†

January 14, 2011

Abstract

We consider the problem of computing numerical invariants of programs by abstract interpretation. Our method eschews two traditional sources of imprecision: (i) the use of widening operators for enforcing convergence within a finite number of iterations (ii) the use of merge operations (often, convex hulls) at the merge points of the control flow graph. It instead computes the least inductive invariant expressible in the domain at a restricted set of program points, and analyzes the rest of the code en bloc. We emphasize that we compute this inductive invariant precisely. For that we extend the strategy improvement algorithm of Gawlitza and Seidl [17]. If we applied their method directly, we would have to solve an exponentially sized system of abstract semantic equations, resulting in memory exhaustion. Instead, we keep the system implicit and discover strategy improvements using SAT modulo real linear arithmetic (SMT). For evaluating strategies we use linear programming. Our algorithm has low polynomial space complexity and performs for contrived examples in the worst case exponentially many strategy improvement steps; this is unsurprising, since we show that the associated abstract reachability problem is Π_2^p-complete.

1 Introduction

Motivation Static program analysis attempts to derive properties about the runtime behavior of a program without running the program. Among interesting properties are the numerical ones: for instance, that a given variable x always has a value in the range $[12, 41]$ when reaching a given program point. An analysis solely based on such interval relations at all program points is known as *interval analysis* [11]. More refined numerical analyses include, for instance, finding for each program point an enclosing polyhedron for the vector of program variables [13]. In addition to obtaining facts about the values of numerical program variables, numerical analyses are used as building blocks for e.g. pointer and shape analyses.

However, by Rice's theorem, only trivial properties can be checked automatically [26]. In order to check non-trivial properties we are usually forced to use *abstractions*. A systematic way for inferring properties automatically w.r.t. a given abstraction

* This work was partially funded by the ANR project "ASOPT".

† Both authors from CNRS (*Centre national de la recherche scientifique*), VERIMAG laboratory. VERIMAG is a joint laboratory of CNRS and Université Joseph Fourier (Grenoble). Email *firstname.lastname*@imag.fr

is given through the *abstract interpretation* framework of Cousot and Cousot [12]. This framework *safely over-approximates* the run-time behavior of a program.

When using the abstract interpretation framework, we usually have two sources of imprecision. The first source of imprecision is the abstraction itself: for instance, if the property to be proved needs a non-convex invariant to be established, and our abstraction can only represent convex sets, then we cannot prove the property. Take for instance the C-code y = 0; **if** (x <= −1 || x >= 1) { **if** (x == 0) y = 1; }. No matter what the values of the variables x and y are before the execution of the above C-code, after the execution the value of y is 0. The invariant $|x| \geq 1$ in the "then" branch is not convex, and its convex hull includes $x = 0$. Any static analysis method that computes a convex invariant in this branch will thus also include $y = 1$. In contrast, our method avoids enforcing convexity, except at the heads of loops.

The second source of imprecision are the safe but imprecise methods that are used for solving the *abstract semantic equations* that describe the abstract semantics: such methods safely over-approximate exact solutions, but do not return exact solutions in all cases. The reason is that we are concerned with abstract domains that contain infinite ascending chains, in particular if we are interested in numerical properties: the complete lattice of all n-dimensional closed real intervals, used for interval analysis, is an example. The traditional methods are based on Kleene fixpoint iteration which (purely applied) is not guaranteed to terminate in interesting cases. In order to enforce termination (for the price of imprecision) traditional methods make use of the widening/narrowing approach of Cousot and Cousot [12]. Grossly, widening extrapolates the first iterations of a sequence to a possible limit, but can easily overshoot the desired result. In order to avoid this, various tricks are used, including "widening up to" [27, Sec. 3.2], "delayed" or with "thresholds" [6]. However, these tricks, although they may help in many practical cases, are easily thwarted. Gopan and Reps [25] proposed "lookahead widening", which discovers new feasible paths and adapts widening accordingly; again this method is no panacea. Furthermore, analyses involving widening are *non-monotonic*: stronger preconditions can lead to weaker invariants being automatically inferred; a rather non-intuitive behaviour. Since our method does not use widening at all, it avoids these problems.

Our Contribution We fight both sources of imprecision noted above:

- In order to improve the precision of the abstraction, we abstract sequences of if-then-else statements without loops en bloc. In the above example, we are then able to conclude that $y \neq 0$ holds. In other words: we abstract sets of states only at the heads of loops, or, more generally, at a cut-set of the control-flow graph (a cut-set is a set of program points such that removing them would cut all loops).
- Our main technical contribution consists of a practical method for precisely computing abstract semantics of affine programs w.r.t. the template linear constraint domains of Sankaranarayanan et al. [42], with sequences of if-then-else statements which do not contain loops abstracted en bloc. Our method is based on a strict generalization of the strategy improvement algorithm of Gawlitza and Seidl [17, 18, 21]. The latter algorithm could be directly applied to the problem we solve in this article, but the size of its input would be

exponential in the size of the program, because we then need to explicitly enumerate all program paths between cut-nodes which do not cross other cut-nodes. In this article, we give an algorithm with low polynomial memory consumption that uses exponential time in the worst case. The basic idea consists in avoiding an explicit enumeration of all paths through sequences of if-then-else-statements which do not contain loops. Instead we use a SAT modulo real linear arithmetic solver for improving the current strategy locally. For evaluating each strategy encountered during the strategy iteration, we use linear programming.

- As a byproduct of our considerations we show that the corresponding abstract reachability problem is Π_2^p-complete. In fact, we show that it is Π_2^p-hard even if the loop invariant being computed consists in a single $x \leq C$ inequality where x is a program variable and C is the parameter of the invariant. Hence, exponential worst-case running-time seems to be unavoidable.

Related Work Recently, several alternative approaches for computing numerical invariants (for instance w.r.t. to template linear constraints) were developed:

Strategy Iteration Strategy iteration (also called *policy iteration*) was introduced by Howard for solving stochastic control problems [29, 40] and is also applied to two-players zero-sum games [28, 39, 45] or min-max-plus systems [7]. Adjé et al. [2], Costan et al. [9], Gaubert et al. [16] developed a strategy iteration approach for solving the abstract semantic equations that occur in static program analysis by abstract interpretation. Their approach can be seen as an alternative to the traditional widening/narrowing approach. The goal of their algorithm is to compute least fixpoints of monotone self-maps f, where $f(x) = \min\{\pi(x) \mid \pi \in \Pi\}$ for all x and Π is a family of self-maps. The assumption is that one can efficiently compute the least fixpoint $\mu\pi$ of π for every $\pi \in \Pi$. The π's are the (min-)strategies. Starting with an arbitrary min-stratgy $\pi^{(0)}$, the min-strategy is successively improved. The sequence $(\pi^{(k)})_k$ of attained min-strategies results in a decreasing sequence $\mu\pi^{(0)} > \mu\pi^{(1)} > \cdots > \mu\pi^{(k)}$ that stabilizes, whenever $\mu\pi^{(k)}$ is a fixpoint of f — not necessarily the least one. However, there are indeed important cases, where minimality of the obtained fixpoint can be guaranteed [1]. Moreover, an important advantage of their algorithm is that it can be stopped at any time with a safe over-approximation. This is in particular interesting if there are infinitely many min-strategies [2]. Costan et al. [9] showed how to use their framework for performing interval analysis without widening. Gaubert et al. [16] extended this work to the following *relational* abstract domains: The *zone domain* [33], the *octagon domain* [34] and in particular the *template linear constraint domains* [42]. Gawlitza and Seidl [17] presented a practical (max-)strategy improvement algorithm for computing least solutions of *systems of rational equations*. Their algorithm enables them to perform a template linear constraint analysis *precisely* — even if the mappings are not non-expansive. This means: Their algorithm always computes *least* solutions of abstract semantic equations — not just some solutions.

Acceleration Techniques Gonnord [23], Gonnord and Halbwachs [24] investigated an improvement of linear relation analysis that consists in computing, when possible, the exact (abstract) effect of a loop. The technique is fully compatible with the use of widening, and whenever it applies, it improves both the precision and the

performance of the analysis. Gawlitza et al. [20], Leroux and Sutre [31] studied cases where interval analysis can be done in polynomial time w.r.t. a uniform cost measure, where memory accesses and arithmetic operations are counted for $\mathcal{O}(1)$.

Quantifier Elimination Recent improvements in SAT/SMT solving techniques have made it possible to perform quantifier elimination on larger formulas [36]. Monniaux [37] developed an analysis method based on quantifier elimination in the theory of rational linear arithmetic. This method targets the same domains as the present article; it however produces a richer result. It can not only compute the least invariant inside the abstract domain of a loop, but also express it as a function of the precondition of the loop; the method outputs the source code of the optimal abstract transformer mapping the precondition to the invariant. Its drawback is its high cost, which makes it practical only on small code fragments; thus, its intended application is *modular analysis*: analyze very precisely small portions of code (functions, modules, nodes of a reactive data-flow program, ...), and use the results for analyzing larger portions, perhaps with another method, including the method proposed in this article.

Mathematical Programming Colón et al. [8], Cousot [10], Sankaranarayanan et al. [41] presented approaches for generating linear invariants that uses *non-linear constraint solving*. Leconte et al. [30] propose a mathematical programming formulation whose constraints define the space of all post-solutions of the abstract semantic equations. The objective function aims at minimizing the result. For programs that use affine assignments and affine guards, only, this yields a *mixed integer linear programming* formulation for interval analysis. The resulting mathematical programming problems can then be solved to guaranteed global optimality by means of general purpose branch-and-bound type algorithms.

2 Basics

Notations $\mathbb{B} = \{0, 1\}$ denotes the set of Boolean values. The set of real numbers is denoted by $\mathbb{R}$. The complete linearly ordered set $\mathbb{R} \cup \{-\infty, \infty\}$ is denoted by $\overline{\mathbb{R}}$. We call two vectors $x, y \in \overline{\mathbb{R}}^n$ *comparable* iff $x \leq y$ or $y \leq x$ holds. For $f : X \to \overline{\mathbb{R}}^m$ with $X \subseteq \overline{\mathbb{R}}^n$, we set $\mathrm{dom}(f) := \{x \in X \mid f(x) \in \mathbb{R}^m\}$ and $\mathrm{fdom}(f) := \mathrm{dom}(f) \cap \mathbb{R}^n$. We denote the i-th row (resp. the j-th column) of a matrix A by $A_{i\cdot}$ (resp. $A_{\cdot j}$). Accordingly, $A_{i\cdot j}$ denotes the component in the i-th row and the j-th column. We also use this notation for vectors and mappings $f : X \to Y^k$.

Assume that a fixed set $\mathbf{X}$ of variables and a domain $\mathbb{D}$ is given. We consider equations of the form $\mathbf{x} = e$, where $\mathbf{x} \in \mathbf{X}$ is a variable and e is an expression over $\mathbb{D}$. A *system* $\mathcal{E}$ of (fixpoint) equations is a finite set $\{\mathbf{x}_1 = e_1, \ldots, \mathbf{x}_n = e_n\}$ of equations, where $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are pairwise distinct variables. We denote the set $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of variables occurring in $\mathcal{E}$ by $\mathbf{X}_\mathcal{E}$. We drop the subscript whenever it is clear from the context.

For a variable assignment $\rho : \mathbf{X} \to \mathbb{D}$, an expression e is mapped to a value $[\![e]\!]\rho$ by setting $[\![\mathbf{x}]\!]\rho := \rho(\mathbf{x})$ and $[\![f(e_1, \ldots, e_k)]\!]\rho := f([\![e_1]\!]\rho, \ldots, [\![e_k]\!]\rho)$, where $\mathbf{x} \in \mathbf{X}$, f is a k-ary operator, for instance $+$, and $e_1, \ldots, e_k$ are expressions. Let $\mathcal{E}$ be a system of equations. We define the unary operator $[\![\mathcal{E}]\!]$ on $\mathbf{X} \to \mathbb{D}$ by setting $([\![\mathcal{E}]\!]\rho)(\mathbf{x}) := [\![e]\!]\rho$ for all $\mathbf{x} = e \in \mathcal{E}$. A solution is a variable assignment ρ such that $\rho = [\![\mathcal{E}]\!]\rho$ holds. The set of solutions is denoted by $\mathbf{Sol}(\mathcal{E})$.

Let $\mathbb{D}$ be a complete lattice. We denote the *least upper bound* and the *greatest lower bound* of a set $X \subseteq \mathbb{D}$ by $\bigvee X$ and $\bigwedge X$, respectively. The least element $\bigvee \emptyset$ (resp. the greatest element $\bigwedge \emptyset$) is denoted by $\bot$ (resp. $\top$). We define the binary operators $\vee$ and $\wedge$ by $x \vee y := \bigvee\{x, y\}$ and $x \wedge y := \bigwedge\{x, y\}$ for all $x, y \in \mathbb{D}$, respectively. For $\square \in \{\vee, \wedge\}$, we will also consider $x_1 \square \cdots \square x_k$ as the application of a k-ary operator. This will cause no problems, since the binary operators $\vee$ and $\wedge$ are associative and commutative. An expression e (resp. an equation $\mathbf{x} = e$) is called *monotone* iff all operators occurring in e are monotone.

The set $\mathbf{X} \to \mathbb{D}$ of all *variable assignments* is a complete lattice. For $\rho, \rho' : \mathbf{X} \to \mathbb{D}$, we write $\rho \lhd \rho'$ (resp. $\rho \rhd \rho'$) iff $\rho(\mathbf{x}) < \rho'(\mathbf{x})$ (resp. $\rho(\mathbf{x}) > \rho'(\mathbf{x})$) holds for all $\mathbf{x} \in \mathbf{X}$. For $d \in \mathbb{D}$, $\underline{d}$ denotes the variable assignment $\{\mathbf{x} \mapsto d \mid \mathbf{x} \in \mathbf{X}\}$. A variable assignment ρ with $\underline{\bot} \lhd \rho \lhd \underline{\top}$ is called *finite*. A pre-solution (resp. post-solution) is a variable assignment ρ such that $\rho \leq [\![\mathcal{E}]\!]\rho$ (resp. $\rho \geq [\![\mathcal{E}]\!]\rho$) holds. The set of all pre-solutions (resp. the set of all post-solutions) is denoted by $\mathbf{PreSol}(\mathcal{E})$ (resp. $\mathbf{PostSol}(\mathcal{E})$). The least fixpoint (resp. the greatest fixpoint) of an operator $f : \mathbb{D} \to \mathbb{D}$ is denoted by μf (resp. νf), provided that it exists. Thus, the least solution (resp. the greatest solution) of a system $\mathcal{E}$ of equations is denoted by $\mu[\![\mathcal{E}]\!]$ (resp. $\nu[\![\mathcal{E}]\!]$), provided that it exists. For a pre-solution ρ (resp. for a post-solution ρ), $\mu_{\geq\rho}[\![\mathcal{E}]\!]$ (resp. $\nu_{\leq\rho}[\![\mathcal{E}]\!]$) denotes the least solution that is greater than or equal to ρ (resp. the greatest solution that is less than or equal to ρ). From Knaster-Tarski's fixpoint theorem we get: Every system $\mathcal{E}$ of monotone equations over a complete lattice has a least solution $\mu[\![\mathcal{E}]\!]$ and a greatest solution $\nu[\![\mathcal{E}]\!]$. Furthermore, $\mu[\![\mathcal{E}]\!] = \bigwedge \mathbf{PostSol}(\mathcal{E})$ and $\nu[\![\mathcal{E}]\!] = \bigvee \mathbf{PreSol}(\mathcal{E})$.

Linear Programming We consider linear programming problems (LP problems for short) of the form $\sup\{c^\top x \mid x \in \mathbb{R}^n, Ax \leq b\}$, where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$ are the inputs. The convex closed polyhedron $\{x \in \mathbb{R}^n \mid Ax \leq b\}$ is called the *feasible space*. The LP problem is called *infeasible* iff the feasible space is empty. An element of the feasible space, is called *feasible solution*. A feasible solution x that maximizes $c^\top x$ is called *optimal solution*.

LP problems can be solved in polynomial time through interior point methods [32, 43]. Note, however, that the running-time then crucially depends on the sizes of occurring numbers. At the danger of an exponential running-time in contrived cases, we can also instead rely on the simplex algorithm: its running-time is *uniform*, i.e., independent of the sizes of occurring numbers (given that arithmetic operations, comparison, storage and retrieval for numbers are counted for $\mathcal{O}(1)$).

SAT modulo real linear arithmetic The set of SAT modulo real linear arithmetic formulas Φ is defined through the grammar $e ::= c \mid x \mid e_1 + e_2 \mid c \cdot e'$, $\Phi ::= a \mid e_1 \leq e_2 \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \wedge \Phi_2 \mid \overline{\Phi'}$. Here, $c \in \mathbb{R}$ is a constant, x is a real valued variable, e, e', e_1, e_2 are real-valued linear expressions, a is a Boolean variable and $\Phi, \Phi', \Phi_1, \Phi_2$ are formulas. An *interpretation* I for a formula Φ is a mapping that assigns a real value to every real-valued variable and a Boolean value to every Boolean variable. We write $I \models \Phi$ for "I is a *model* of Φ", i.e., $[\![c]\!]I = c$, $[\![x]\!] = I(x)$, $[\![e_1 + e_2]\!]I = [\![e_1]\!]I + [\![e_2]\!]I$, $[\![c \cdot e']\!]I = c \cdot [\![e']\!]I$, and:

$$I \models a \iff I(a) = 1 \qquad\qquad I \models e_1 \leq e_2 \iff [\![e_1]\!]I \leq [\![e_2]\!]I$$

$$I \models \Phi_1 \vee \Phi_2 \iff I \models \Phi_1 \text{ or } I \models \Phi_2 \quad I \models \Phi_1 \wedge \Phi_2 \iff I \models \Phi_1 \text{ and } I \models \Phi_2$$
$$I \models \overline{\Phi'} \iff I \not\models \Phi'$$

A formula is called *satisfiable* iff it has a model. The problem of deciding, whether or not a given SAT modulo real linear arithmetic formula is satisfiable, is NP-complete. There nevertheless exist efficient solver implementations for this decision problem [15].

In order to simplify notations we also allow matrices, vectors, the operations $\geq$, $<, >, \neq, =$, and the Boolean constants 0 and 1 to occur.

Collecting and Abstract Semantics The programs that we consider in this article use real-valued variables $x_1, \ldots, x_n$. Accordingly, we denote by $x = (x_1, \ldots, x_n)^\top$ the vector of all program variables. For simplicity, we only consider elementary statements of the form $x := Ax + b$, and $Ax \leq b$, where $A \in \mathbb{R}^{n \times n}$ (resp. $\mathbb{R}^{k \times n}$), $b \in \mathbb{R}^n$ (resp. $\mathbb{R}^k$), and $x \in \mathbb{R}^n$ denotes the vector of all program variables. Statements of the form $x := Ax + b$ are called *(affine) assignments*. Statements of the form $Ax \leq b$ are called *(affine) guards*. Additionally, we allow statements of the form $s_1; \cdots ; s_k$ and $s_1 \mid \cdots \mid s_k$, where $s_1, \ldots, s_k$ are statements. The operator ; binds tighter than the operator |, and we consider ; and | to be right-associative, i.e., $s_1 \mid s_2 \mid s_3$ stands for $s_1 \mid (s_2 \mid s_3)$, and $s_1; s_2; s_3$ stands for $s_1; (s_2; s_3)$. The set of statements is denoted by **Stmt**. A statement of the form $s_1 \mid \cdots \mid s_k$, where s_i does not contain the operator | for all $i = 1, \ldots, k$, is called *merge-simple*. A merge-simple statement s that does not use the | operator at all is called *sequential*. A statement is called *elementary* iff it neither contains the operator | nor the operator ;.

The *collecting semantics* $[\![s]\!] : 2^{\mathbb{R}^n} \to 2^{\mathbb{R}^n}$ of a statement $s \in \mathbf{Stmt}$ is defined by

$$[\![x := Ax + b]\!]X := \{Ax + b \mid x \in X\}, \qquad [\![Ax \leq b]\!]X := \{x \in X \mid Ax \leq b\},$$
$$[\![s_1; \cdots ; s_k]\!] := [\![s_k]\!] \circ \cdots \circ [\![s_1]\!] \qquad [\![s_1 \mid \cdots \mid s_k]\!]X := [\![s_1]\!]X \cup \cdots \cup [\![s_k]\!]X$$

for $X \subseteq \mathbb{R}^n$. Note that the operators ; and | are associative, i.e., $[\![(s_1; s_2); s_3]\!] = [\![s_1; (s_2; s_3)]\!]$ and $[\![(s_1 \mid s_2) \mid s_3]\!] = [\![s_1 \mid (s_2 \mid s_3)]\!]$ hold for all statements s_1, s_2, s_3.

An *(affine) program* G is a triple $(N, E, \mathbf{st})$, where N is a finite set of *program points*, $E \subseteq N \times \mathbf{Stmt} \times N$ is a finite set of control-flow edges, and $\mathbf{st} \in N$ is the *start program point*. As usual, the *collecting semantics* V of a program $G = (N, E, \mathbf{st})$ is the least solution of the following constraint system:

$$\mathbf{V}[\mathbf{st}] \supseteq \mathbb{R}^n \qquad \mathbf{V}[v] \supseteq [\![s]\!](\mathbf{V}[u]) \quad \text{for all } (u, s, v) \in E$$

Here, the variables $\mathbf{V}[v]$, $v \in N$ take values in $2^{\mathbb{R}^n}$. The components of the collecting semantics V are denoted by $V[v]$ for all $v \in N$.

Let $\mathbb{D}$ be a complete lattice (for instance the complete lattice of all n-dimensional closed real intervals). Let the partial order of $\mathbb{D}$ be denoted by $\leq$. Assume that $\alpha : 2^{\mathbb{R}^n} \to \mathbb{D}$ and $\gamma : \mathbb{D} \to 2^{\mathbb{R}^n}$ form a Galois connection, i.e., for all $X \subseteq \mathbb{R}^n$ and all $d \in \mathbb{D}$, $\alpha(X) \leq d$ iff $X \subseteq \gamma(d)$. The *abstract semantics* $[\![s]\!]^\sharp : \mathbb{D} \to \mathbb{D}$ of a statement s is defined by $[\![s]\!]^\sharp := \alpha \circ [\![s]\!] \circ \gamma$. The *abstract semantics* $V^\sharp$ of an affine program $G = (N, E, \mathbf{st})$ is the least solution of the following constraint system:

$$\mathbf{V}^\sharp[\mathbf{st}] \geq \alpha(\mathbb{R}^n) \qquad \mathbf{V}^\sharp[v] \geq [\![s]\!]^\sharp(\mathbf{V}^\sharp[u]) \quad \text{for all } (u, s, v) \in E$$

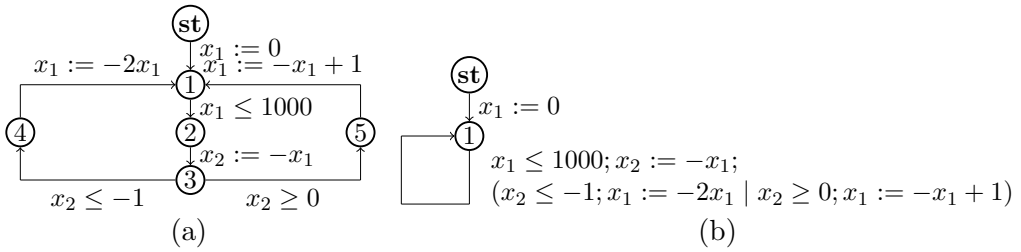

Figure 1:

Here, the variables $\mathbf{V}^\sharp[v]$, $v \in N$ take values in $\mathbb{D}$. The components of the abstract semantics $V^\sharp$ are denoted by $V^\sharp[v]$ for all $v \in N$. The abstract semantics $V^\sharp$ safely over-approximates the collecting semantics V, i.e., $\gamma(V^\sharp[v]) \supseteq V[v]$ for all $v \in N$.

Using Cut-Sets to improve Precision Usually, only sequential statements (these statements correspond to *basic blocks*) are allowed in control flow graphs. However, given a cut-set C, one can systematically transform any control flow graph G into an equivalent control flow graph G' of our form (up to the fact that G' has fewer program points than G) with increased precision of the abstract semantics. However, for the sake of simplicity, we do not discuss these aspects in detail. Instead, we consider an example:

Example 1 (Using Cut-Sets to improve Precision). As a running example throughout the present article we use the following C-code:

```
int x_1, x_2; x_1 = 0; while (x_1 <= 1000) { x_2 = -x_1;
  if (x_2 < 0) x_1 = -2 * x_1; else x_1 = -x_1 + 1; }
```

This C-code is abstracted through the affine program $G_1 = (N_1, E_1, \mathbf{st})$ which is shown in Figure 1.(a). However, it is unnecessary to apply abstraction at every program point; it suffices to apply abstraction at a cut-set of G_1. Since all loops contain program point 1, a cut-set of G_1 is $\{1\}$. Equivalent to applying abstraction only at program point 1 is to rewrite the control-flow graph w.r.t. the cut-set $\{1\}$ into a control-flow graph G equivalent w.r.t. the collecting semantic. The result of this transformation is drawn in Figure 1.(b). This means: the affine program for the above C-code is $G = (N, E, \mathbf{st})$, where $N = \{\mathbf{st}, 1\}, E = \{(\mathbf{st}, x_1 := 0, 1), (1, s, 1)\}$, and

$$s' = x_1 \leq 1000; x_2 := -x_1 \qquad s_1 = x_2 \leq -1; x_1 := -2x_1$$
$$s_2 = -x_2 \leq 0; x_1 := -x_1 + 1 \qquad s = s'; (s_1 \mid s_2)$$

Let V_1 denote the collecting semantics of G_1 and V denote the collecting semantics of G. G_1 and G are equivalent in the following sense: $V[v] = V_1[v]$ holds for all program points $v \in N$. W.r.t. the abstract semantics, G is, is we will see, strictly more precise than G_1. In general we at least have $V^\sharp[v] \subseteq V_1^\sharp.[v]$ for all program points $v \in N$. This is independent of the abstract domain.[1] □

[1] We assume that we have given a Galois-connection and thus in particular monotone best abstract transformers.

Template Linear Constraints In the present article we restrict our considerations to *template linear constraint domains* [42]. Assume that we are given a fixed *template constraint matrix* $T \in \mathbb{R}^{m \times n}$. The template linear constraint domain is $\overline{\mathbb{R}}^m$. As shown by Sankaranarayanan et al. [42], the concretization $\gamma : \overline{\mathbb{R}}^m \to 2^{\mathbb{R}^n}$ and the abstraction $\alpha : 2^{\mathbb{R}^n} \to \overline{\mathbb{R}}^m$, which are defined by

$$\begin{aligned} \gamma(d) &:= \{x \in \mathbb{R}^n \mid Tx \leq d\} && \forall d \in \overline{\mathbb{R}}^m, \\ \alpha(X) &:= \bigwedge\{d \in \overline{\mathbb{R}}^m \mid \gamma(d) \supseteq X\} && \forall X \subseteq \mathbb{R}^n, \end{aligned}$$

form a Galois connection. The template linear constraint domains contain *intervals*, *zones*, and *octagons*, with appropriate choices of the template constraint matrix [42].

In a first stage we restrict our considerations to sequential and merge-simple statements. Even for these statements we avoid unnecessary imprecision, if we abstract such statements en bloc instead of abstracting each elementary statement separately:

Example 2. In this example we use the interval domain as abstract domain, i.e., our complete lattice consists of all n-dimensional closed real intervals. Our affine program will use 2 variables, i.e., $n = 2$. The complete lattice of all 2-dimensional closed real intervals can be specified through the template constraint matrix $T = \begin{pmatrix} -I & I \end{pmatrix}^\top \in \mathbb{R}^{4 \times 2}$, where I denotes the identity matrix. Consider the statements $s_1 = x_2 := x_1$, $s_2 = x_1 := x_1 - x_2$, and $s = s_1; s_2$ and the abstract value $I = [0, 1] \times \mathbb{R}$ (a 2-dimensional closed real interval). The interval I can w.r.t. T be identified with the abstract value $(0, \infty, 1, \infty)^\top$. More generally, w.r.t. T every 2-dimensional closed real interval $[l_1, u_1] \times [l_2, u_2]$ can be identified with the abstract value $(-l_1, -l_2, u_1, u_2)^\top$. If we abstract each elementary statement separately, then we in fact use $[\![s_2]\!]^\sharp \circ [\![s_1]\!]^\sharp$ instead of $[\![s]\!]^\sharp$ to abstract the collecting semantics $[\![s]\!]$ of the statement $s = s_1; s_2$. The following calculation shows that this can be important: $[\![s]\!]^\sharp I = [0, 0] \times [0, 1] \neq [-1, 1] \times [0, 1] = [\![s_2]\!]^\sharp([0, 1] \times [0, 1]) = ([\![s_2]\!]^\sharp \circ [\![s_1]\!]^\sharp)I$. The imprecision is caused by the additional abstraction. We lose the information that the values of the program variables x_1 and x_2 are equal after executing the first statement. □

Another possibility for avoiding unnecessary imprecision in the above example would consist in adding additional rows to the template constraint matrix. Although this works for the above example, it does not work in general, since still only convex sets can be described, but sometimes non-convex sets are required (cf. with the example in the introduction).

Provided that s is a merge-simple statement, $[\![s]\!]^\sharp d$ can be computed in polynomial time through linear programming:

Lemma 3 (Merge-Simple Statements)**.** *Let s be a merge-simple statement and $d \in \overline{\mathbb{R}}^m$. Then $[\![s]\!]^\sharp d$ can be computed in polynomial time through linear programming.* □

However, the situation for arbitrary statements is significantly more difficult, since, by reducing SAT to the corresponding decision problem, we can show the following:

Lemma 4. *The problem of deciding, whether or not, for a given template constraint matrix T, and a given statement s, $[\![s]\!]^\sharp \underline{\infty} > \underline{-\infty}$ holds, is NP-complete.*

Before proving the above lemma, we introduce $\vee$-strategies for statements as follows:

Definition 1 ($\vee$-Strategies for Statements)**.** A $\vee$-strategy σ for a statement s is a function that maps every position of a $|$-statement, (a statement of the form $s_0 \mid s_1$) within s to 0 or 1. The application $s\sigma$ of a $\vee$-strategy σ to a statement s is inductively defined by $s\sigma = s$, $(s_0 \mid s_1)\sigma = s_{\sigma(\mathrm{pos}(s_0|s_1))}\sigma$, and $(s_0; s_1)\sigma = (s_0\sigma; s_1\sigma)$, where s is an elementary statement, and s_0, s_1 are arbitrary statements. For all occurrences s', $\mathrm{pos}(s')$ denotes the position of s', i.e., $\mathrm{pos}(s')$ identifies the occurrence. □

Proof. Firstly, we show containment in NP. Assume $[\![s]\!]^\sharp \underline{\infty} > \underline{-\infty}$. There exists some k such that the k-th component of $[\![s]\!]^\sharp \underline{\infty}$ is greater than $-\infty$. We choose k non-deterministically. There exists a $\vee$-strategy σ for s such that the k-th component of $[\![s\sigma]\!]^\sharp \underline{\infty}$ equals the k-th component of $[\![s]\!]^\sharp \underline{\infty}$. We choose such a $\vee$-strategy non-deterministically. By Lemma 3, we can check in polynomial time, whether the k-th component of $[\![s\sigma]\!]^\sharp \underline{\infty}$ is greater than $-\infty$. If this is fulfilled, we accept.

In order to show NP-hardness, we reduce the NP-hard problem SAT to our problem. Let Φ be a propositional formula with n variables. W.l.o.g. we assume that Φ is in normal form, i.e., there are no negated sub-formulas that contain $\wedge$ or $\vee$. We define the statement $s(\Phi)$ that uses the variables of Φ as program variables inductively by $s(z) := z = 1$, $s(\overline{z}) := z = 0$, $s(\Phi_1 \wedge \Phi_2) := s(\Phi_1); s(\Phi_2)$, and $s(\Phi_1 \vee \Phi_2) := s(\Phi_1) \mid s(\Phi_2)$, where z is a variable of Φ, and Φ_1, Φ_2 are formulas. Here, the statement $Ax = b$ is an abbreviation for the statement $Ax \leq b; -Ax \leq -b$. The formula Φ is satisfiable iff $[\![s(\Phi)]\!]\mathbb{R}^n \neq \emptyset$ holds. Moreover, even if we just use the interval domain, $[\![s(\Phi)]\!]\mathbb{R}^n \neq \emptyset$ holds iff $[\![s(\Phi)]\!]^\sharp \underline{\infty} > \underline{-\infty}$ holds. Thus, Φ is satisfiable iff $[\![s(\Phi)]\!]^\sharp \underline{\infty} > \underline{-\infty}$ holds. □ □

Obviously, $[\![(s_1 \mid s_2); s]\!] = [\![s_1; s \mid s_2; s]\!]$ and $[\![s; (s_1 \mid s_2)]\!] = [\![s; s_1 \mid s; s_2]\!]$ for all statements s, s_1, s_2. We can transform any statement s into an equivalent merge-simple statement s' using these rules. We denote the merge-simple statement s' that is obtained from an arbitrary statement s by applying the above rules in some canonical way by $[s]$. Intuitively, $[s]$ is an explicit enumeration of all paths through the statement s.

Lemma 5. *For every statement s, $[s]$ is merge-simple, and $[\![s]\!] = [\![[s]]\!]$. The size of $[s]$ is at most exponential in the size of s.* □

However, in the worst case, the size of $[s]$ is exponential in the size of s. For the statement $s = (s_1^{(1)} \mid s_1^{(2)}); \cdots ; (s_k^{(1)} \mid s_k^{(2)})$, for instance, we get $[s] = |_{(a_1,\ldots,a_k)\in\{1,2\}^k} s_1^{(a_1)}; \cdots ; s_k^{(a_k)}$. After replacing all statements s with $[s]$ it is in principle possible to use the methods of Gawlitza and Seidl [17] in order to compute the abstract semantics $V^\sharp$ precisely. Because of the exponential blowup, however, this method would be impractical in most cases. [2]

Our new method that we are going to present avoids this exponential blowup: instead of enumerating all program paths, we shall visit them only as needed. Guided by a SAT modulo real linear arithmetic solver, our method selects a path through

[2] Note that we cannot expect a polynomial-time algorithm, because of Lemma 4: even without loops, abstract reachability is NP-hard. Even if all statements are merge-simple, we cannot expect a polynomial-time algorithm, since the problem of computing the winning regions of parity games is polynomial-time reducible to abstract reachability [19].

s only when it is *locally profitable* in some sense. In the worst case, an exponential number of paths may be visited (Section 7); but one can hope that this does not happen in many practical cases, in the same way that SAT and SMT solving perform well on many practical cases even though they in principle may visit an exponential number of cases.

Abstract Semantic Equations The first step of our method consists of rewriting our program analysis problem into a *system of abstract semantic equations* that is interpreted over the reals. For that, let $G = (N, E, \mathbf{st})$ be an affine program and $V^\sharp$ its abstract semantics. We define the system $\mathcal{C}(G)$ of *abstract semantic inequalities* to be the smallest set of inequalities that fulfills the following constraints:

- $\mathcal{C}$ contains the inequality $\mathbf{x}_{\mathbf{st},i} \geq \alpha_{i\cdot}(\mathbb{R}^n)$ for every $i \in \{1, \ldots, m\}$.
- $\mathcal{C}$ contains the inequality $\mathbf{x}_{v,i} \geq [\![s]\!]^\sharp_{i\cdot}(\mathbf{x}_{u,1}, \ldots, \mathbf{x}_{u,m})$ for every control-flow edge $(u, s, v) \in E$ and every $i \in \{1, \ldots, m\}$.

We define the system $\mathcal{E}(G)$ of *abstract semantic equations* by $\mathcal{E}(G) := \mathcal{E}(\mathcal{C}(G))$. Here, for a system $\mathcal{C}' = \{\mathbf{x}_1 \geq e_{1,1}, \ldots, \mathbf{x}_1 \geq e_{1,k_1}, \ldots, \mathbf{x}_n \geq e_{n,1}, \ldots, \mathbf{x}_n \geq e_{n,k_n}\}$ of inequalities, $\mathcal{E}(\mathcal{C}')$ is the system $\mathcal{E}(\mathcal{C}') = \{\mathbf{x}_1 = e_{1,1} \vee \cdots \vee e_{1,k_1}, \ldots, \mathbf{x}_n = e_{n,1} \vee \cdots \vee e_{n,k_n}\}$ of equations. The system $\mathcal{E}(G)$ of abstract semantic equations captures the abstract semantics $V^\sharp$ of G:

Lemma 6. *$(V^\sharp[v])_{i\cdot} = \mu[\![\mathcal{E}(G)]\!](\mathbf{x}_{v,i})$ for all program points v, $i \in \{1, \ldots, m\}$.* □

Example 7 (Abstract Semantic Equations). We again consider the program G of Example 1. Assume that the template constraint matrix $T \in \mathbb{R}^{2\times 2}$ is given by $T_{1\cdot} = (1, 0)$ and $T_{2\cdot} = (-1, 0)$. Let $V^\sharp$ denote the abstract semantics of G. Then $V^\sharp[1] = (2001, 2000)^\top$. $\mathcal{E}(G)$ consists of the following abstract semantic equations:

$$\begin{aligned} \mathbf{x}_{\mathbf{st},1} &= \infty & \mathbf{x}_{1,1} &= [\![x_1 := 0]\!]^\sharp_{1\cdot}(\mathbf{x}_{\mathbf{st},1}, \mathbf{x}_{\mathbf{st},2}) \vee [\![s]\!]^\sharp_{1\cdot}(\mathbf{x}_{1,1}, \mathbf{x}_{1,2}) \\ \mathbf{x}_{\mathbf{st},2} &= \infty & \mathbf{x}_{1,2} &= [\![x_1 := 0]\!]^\sharp_{2\cdot}(\mathbf{x}_{\mathbf{st},1}, \mathbf{x}_{\mathbf{st},2}) \vee [\![s]\!]^\sharp_{2\cdot}(\mathbf{x}_{1,1}, \mathbf{x}_{1,2}) \end{aligned}$$

As stated by Lemma 6, we have $(V^\sharp[1])_{1\cdot} = \mu[\![\mathcal{E}(G)]\!](\mathbf{x}_{1,1}) = 2001$, and $(V^\sharp[1])_{2\cdot} = \mu[\![\mathcal{E}(G)]\!](\mathbf{x}_{1,2}) = 2000$. □

3 A Lower Bound on the Complexity

In this section we show that the problem of computing abstract semantics of affine programs w.r.t. the interval domain is Π^p_2-hard. Π^p_2-hard problems are conjectured to be harder than both NP-complete and co-NP-complete problems. For further information regarding the polynomial-time hierarchy see e.g. Stockmeyer [44].

Theorem 8. *The problem of deciding, whether, for a given program G, a given template constraint matrix T, and a given program point v, $V^\sharp[v] > \underline{-\infty}$ holds, is Π^p_2-hard.*

Proof. We reduce the Π^p_2-complete problem of deciding the truth of a $\forall\exists$ propositional formula [46] to our problem. Let $\Phi = \forall x_1, \ldots, x_n.\exists y_1, \ldots, y_m.\Phi'$ be a formula without free variables, where Φ' is a propositional formula. We consider the affine

program $G = (N, E, \mathbf{st})$, with program variables $x, x', x_1, \ldots, x_n, y_1, \ldots, y_m$, where $N = \{\mathbf{st}, 1, 2\}$, and $E = \{(\mathbf{st}, x := 0, 1), (1, s, 1), (1, x \geq 2^n, 2)\}$ with

$$\begin{aligned} s \;=\; & x' := x; (x' \geq 2^{n-1}; x' := x' - 2^{n-1}; x_n := 1 \mid x' \leq 2^{n-1} - 1; x_n := 0); \cdots \\ & (x' \geq 2^{1-1}; x' := x' - 2^{1-1}; x_1 := 1 \mid x' \leq 2^{1-1} - 1; x_1 := 0); \\ & s(\Phi');\ x := x + 1 \end{aligned}$$

The statement $s(\Phi')$ is defined as in the proof of Lemma 4.

In intuitive terms: this program initializes x to 0. Then, it enters a loop: it computes into $x_1, \ldots, x_n$ the binary decomposition of x, then it attempts to nondeterministically choose $y_1, \ldots, y_m$ so that ϕ' is true. If this is possible, it increments x by one and loops. Otherwise, it just loops. Thus, there is a terminating computations iff Φ holds.

Then Φ holds iff $V[2] \neq \emptyset$. For the abstraction, we consider the interval domain. By considering the Kleene-Iteration, it is easy to see that $V[2] \neq \emptyset$ holds iff $V^\sharp[2] > \underline{-\infty}$ holds. Thus Φ holds iff $V^\sharp[2] > \underline{-\infty}$ holds. □ □

4 Determining Improved Strategies

In this section we develop a method for computing local improvements of strategies through solving SAT modulo real linear arithmetic formulas.

In order to decide, whether or not, for a given statement s, a given $j \in \{1, \ldots, m\}$, a given c, and a given $d \in \overline{\mathbb{R}}^m$, $[\![s]\!]^\sharp_j.d > c$ holds, we construct the following SAT modulo real linear arithmetic formula (we use existential quantifiers to improve readability):

$$\begin{aligned} \Phi(s, d, j, c) &:\equiv \exists v \in \mathbb{R} \,.\, \Phi(s, d, j) \wedge v > c \\ \Phi(s, d, j) &:\equiv \exists x \in \mathbb{R}^n, x' \in \mathbb{R}^n \,.\, Tx \leq d \wedge \Phi(s) \wedge v = T_j.x' \end{aligned}$$

Here, $\Phi(s)$ is a formula that relates every $x \in \mathbb{R}^n$ with all elements from the set $[\![s]\!]\{x\}$. It is defined inductively over the structure of s as follows:

$$\begin{aligned} \Phi(x := Ax + b) &:\equiv x' = Ax + b \\ \Phi(Ax \leq b) &:\equiv Ax \leq b \wedge x' = x \\ \Phi(s_1; s_2) &:\equiv \exists x'' \in \mathbb{R}^n \,.\, \Phi(s_1)[x''/x'] \wedge \Phi(s_2)[x''/x] \\ \Phi(s_1 \mid s_2) &:\equiv (\overline{a_{\mathrm{pos}(s_1|s_2)}} \wedge \Phi(s_1)) \vee (a_{\mathrm{pos}(s_1|s_2)} \wedge \Phi(s_2)) \end{aligned}$$

Here, for every position p of a subexpression of s, a_p is a Boolean variable. Let $\mathrm{Pos}_|(s)$ denote the set of all positions of $|$-subexpressions of s. The set of free variables of the formula $\Phi(s)$ is $\{x, x'\} \cup \{a_p \mid p \in \mathrm{Pos}_|(s)\}$. A valuation for the variables from the set $\{a_p \mid p \in \mathrm{Pos}_|(s)\}$ describes a path through s. We have:

Lemma 9. *$[\![s]\!]^\sharp_j.d > c$ holds iff $\Phi(s, d, j, c)$ is satisfiable.* □

Our next goal is to compute a $\vee$-strategy σ for s such that $[\![s\sigma]\!]^\sharp_j.d > c$ holds, provided that $[\![s]\!]^\sharp_j.d > c$ holds. Let s be a statement, $d \in \overline{\mathbb{R}}^m$, $j \in \{1, \ldots, m\}$, and $c \in \mathbb{R}$. Assume that $[\![s]\!]^\sharp_j.d > c$ holds. By Lemma 9, there exists a model M of $\Phi(s, d, j, c)$. We define the $\vee$-strategy σ_M for s by $\sigma_M(p) := M(a_p)$ for all $p \in \mathrm{Pos}_|(s)$. By again applying Lemma 9, we get $[\![s\sigma]\!]^\sharp_j.d > c$. Summarizing we have:

$$
\begin{aligned}
\Phi(s,(0,0)^\top,1,0) &\equiv \exists v \in \mathbb{R} \,.\, \Phi(s,(0,0)^\top,1) \wedge v > 0\\
\Phi(s,(0,0)^\top,1) &\equiv \exists x \in \mathbb{R}^2, x' \in \mathbb{R}^2 \,.\, x_{1.} \leq 0 \wedge -x_{1.} \leq 0 \wedge \Phi(s) \wedge v = x'_{1.}\\
\Phi(s') &\equiv \exists x'' \in \mathbb{R}^2 \,.\, x_{1.} \leq 1000 \wedge x''_{1.} = x_{1.} \wedge x''_{2.} = x_{2.} \wedge x'_{1.} = x''_{1.} \wedge x'_{2.} = -x''_{1.}\\
&\equiv x_{1.} \leq 1000 \wedge x'_{1.} = x_{1.} \wedge x'_{2.} = -x_{1.}\\
\Phi(s_1) &\equiv \exists x'' \in \mathbb{R}^2 \,.\, x_{2.} \leq -1 \wedge x''_{1.} = x_{1.} \wedge x''_{2.} = x_{2.} \wedge x'_{1.} = -2x''_{1.} \wedge x'_{2.} = x''_{2.}\\
&\equiv x_{2.} \leq -1 \wedge x'_{1.} = -2x_{1.} \wedge x'_{2.} = x_{2.}\\
\Phi(s_2) &\equiv \exists x'' \in \mathbb{R}^2 \,.\, -x_{2.} \leq 0 \wedge x''_{1.} = x_{1.} \wedge x''_{2.} = x_{2.} \wedge x'_{1.} = -x''_{1.} + 1 \wedge x'_{2.} = x''_{2.}\\
&\equiv x_{2.} \leq 0 \wedge x'_{1.} = -x_{1.} + 1 \wedge x'_{2.} = x_{2.}\\
\Phi(s_1 \mid s_2) &\equiv (\overline{a_1} \wedge \Phi(s_1)) \vee (a_1 \wedge \Phi(s_2)) \equiv (\overline{a_1} \wedge x_{2.} \leq -1 \wedge x'_{1.} = -2x_{1.} \wedge x'_{2.} = x_{2.})\\
&\qquad \vee (a_1 \wedge x_{2.} \leq 0 \wedge x'_{1.} = -x_{1.} + 1 \wedge x'_{2.} = x_{2.})\\
\Phi(s) &\equiv \exists x'' \in \mathbb{R}^2 \,.\, \Phi(s')[x''/x'] \wedge \Phi(s_1 \mid s_2)[x''/x]\\
&\equiv x_{1.} \leq 1000 \wedge ((\overline{a_1} \wedge -x_{1.} \leq -1 \wedge x'_{1.} = -2x_{1.} \wedge x'_{2.} = -x_{1.})\\
&\qquad \vee (a_1 \wedge -x_{1.} \leq 0 \wedge x'_{1.} = -x_{1.} + 1 \wedge x'_{2.} = -x_{1.}))
\end{aligned}
$$

Figure 2: Formula for Example 11

Lemma 10. *By solving the SAT modulo real linear arithmetic formula $\Phi(s,d,j,c)$ that can be obtained from s in linear time, we can decide, whether or not $[\![s]\!]^\sharp_{j.}d > c$ holds. From a model M of this formula, we can obtain a $\vee$-strategy σ_M for s such that $[\![s\sigma_M]\!]^\sharp_{j.}d > c$ holds in linear time.* □

Example 11. We again continue Example 1 and 7. We want to know, whether $[\![s]\!]^\sharp_{1.}(0,0)^\top > 0$ holds. For that we compute a model of the formula $\Phi(s,(0,0)^\top,1,0)$ which is written down in Figure 2. $M = \{a_1 \mapsto 1\}$ is a model of the formula $\Phi(s,(0,0)^\top,1,0)$. Thus, we have $0 < [\![s\sigma_M]\!]^\sharp_{1.}(0,0)^\top = [\![s';s_2]\!]^\sharp_{1.}(0,0)^\top$ by Lemma 10. □

It remains to compute a model of $\Phi(s,d,j,c)$. Most of the state-of-the-art SMT solvers, as for instance Yices [14, 15], support the computation of models directly; if unsupported, one can compute the model using standard self-reduction techniques.

The semantic equations we are concerned with in the present article have the form $\mathbf{x} = e_1 \vee \cdots \vee e_k$, where each expression e_i, $i = 1,\ldots,k$ is either a constant or an expression of the form $[\![s]\!]^\sharp_{j.}(\mathbf{x}_1,\ldots,\mathbf{x}_m)$. We now extent our notion of $\vee$-strategies in order to deal with the occurring right-hand sides:

Definition 2 ($\vee$-Strategies)**.** The $\vee$-strategy for all constants is the 0-tuple $()$. The application $c()$ of $()$ to a constant $c \in \overline{\mathbb{R}}$ is defined by $c() := c$ for all $c \in \overline{\mathbb{R}}$. A $\vee$-strategy σ for an expression $[\![s]\!]^\sharp_{j.}(\mathbf{x}_1,\ldots,\mathbf{x}_m)$ is a $\vee$-strategy for s. The application $([\![s]\!]^\sharp_{j.}(\mathbf{x}_1,\ldots,\mathbf{x}_m))\sigma$ of σ to $[\![s]\!]^\sharp_{j.}(\mathbf{x}_1,\ldots,\mathbf{x}_m)$ is defined by $([\![s]\!]^\sharp_{j.}(\mathbf{x}_1,\ldots,\mathbf{x}_m))\sigma := [\![s\sigma]\!]^\sharp_{j.}(\mathbf{x}_1,\ldots,\mathbf{x}_m)$. A $\vee$-strategy for an expression $e = e_0 \vee e_1,$, where, for each $i \in \{0,1\}$, e_i is either a constant or an expression of the form $[\![s]\!]^\sharp_{j.}(\mathbf{x}_1,\ldots,\mathbf{x}_m)$, is a pair (p,σ), where $p \in \{0,1\}$ and σ is a $\vee$-strategy for e_p. The application $e(p,\sigma)$ of (p,σ) to $e = e_0 \vee e_1$ is defined by $e(p,\sigma) = e_p\sigma$. A $\vee$-strategy σ for a system $\mathcal{E} = \{\mathbf{x}_1 = e_1,\ldots,\mathbf{x}_n = e_n\}$ of abstract semantic equations is a mapping $\{\mathbf{x}_i \mapsto \sigma_i \mid i = 1,\ldots,n\}$, where σ_i is a $\vee$-strategy for e_i for all $i = 1,\ldots,n$. We set $\mathcal{E}(\sigma) := \{\mathbf{x}_1 = e_1(\sigma(\mathbf{x}_1)),\ldots,\mathbf{x}_n = e_n(\sigma(\mathbf{x}_n))\}$. □

Using the same ideas as above, we can prove the following lemma which finally enables us to use a SAT modulo real linear arithmetic solver for improving $\vee$-strategies for systems of abstract semantic equations locally.

Lemma 12. *Let $\mathbf{x} = e$ be an abstract semantic equation, ρ a variable assignment, and $c \in \overline{\mathbb{R}}$. By solving a SAT modulo real linear arithmetic formula that can be obtained from e, ρ and c in linear time, we can decide, whether or not $[\![e]\!]\rho > c$ holds. From a model M of this formula, we can in linear time obtain a $\vee$-strategy σ_M for e such that $[\![e\sigma_M]\!]\rho > c$ holds.* □

5 Solving Systems of Concave Equations

In order to solve systems of abstract semantic equations (see the end of Section 2) we generalize the $\vee$-strategy improvement algorithm of Gawlitza and Seidl [21] as follows:

Concave Functions A set $X \subseteq \mathbb{R}^n$ is called *convex* iff $\lambda x + (1-\lambda)y \in X$ holds for all $x, y \in X$ and all $\lambda \in [0,1]$. A mapping $f : X \to \mathbb{R}^m$ with $X \subseteq \mathbb{R}^n$ convex is called *convex* (resp. *concave*) iff $f(\lambda x + (1-\lambda)y) \leq$ (resp. $\geq$) $\lambda f(x) + (1-\lambda)f(y)$ holds for all $x, y \in X$ and all $\lambda \in [0,1]$. Note that f is concave iff $-f$ is convex. Note also that f is convex (resp. concave) iff $f_{i\cdot}$ is convex (resp. concave) for all $i = 1, \ldots, m$.

We extend the notion of convexity/concavity from $\mathbb{R}^n \to \mathbb{R}^m$ to $\overline{\mathbb{R}}^n \to \overline{\mathbb{R}}^m$ as follows: Let $f : \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}^m$, and $I : \{1, \ldots, n\} \to \{-\infty, \mathsf{id}, \infty\}$. Here, $-\infty$ denotes the function that assigns $-\infty$ to every argument, id denotes the identity function, and ∞ denotes the function that assigns ∞ to every argument. We define the mapping $f^{(I)} : \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}^m$ by $f^{(I)}(x_1, \ldots, x_n) := f(I(1)(x_1), \ldots, I(n)(x_n))$ for all $x_1, \ldots, x_n \in \overline{\mathbb{R}}$. A mapping $f : \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}^m$ is called *concave* iff $f_{i\cdot}$ is continuous on $\{x \in \overline{\mathbb{R}}^n \mid f_{i\cdot}(x) > -\infty\}$ for all $i \in \{1, \ldots, m\}$, and the following conditions are fulfilled for all $I : \{1, \ldots, n\} \to \{-\infty, \mathsf{id}, \infty\}$:

1. $\mathrm{fdom}(f^{(I)})$ is convex.
2. $f^{(I)}|_{\mathrm{fdom}(f^{(I)})}$ is concave.
3. For all $i \in \{1, \ldots, m\}$ the following holds: If there exists some $y \in \mathbb{R}^n$ such that $f_{i\cdot}^{(I)}(y) \in \mathbb{R}$, then $f_{i\cdot}^{(I)}(x) < \infty$ for all $x \in \mathbb{R}^n$.

A mapping $f : \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}^m$ is called *convex* iff $-f$ is concave. In the following we are only concerned with mappings $f : \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}^m$ that are monotone and concave.

We slightly extend the definition of concave equations of Gawlitza and Seidl [21]:

Definition 3 (Concave Equations)**.** An expression e (resp. equation $\mathbf{x} = e$) over $\overline{\mathbb{R}}$ is called *basic concave expression* (resp. *basic concave equation*) iff $[\![e]\!]$ is monotone and concave. An expression e (resp. equation $\mathbf{x} = e$) over $\overline{\mathbb{R}}$ is called *concave* iff $e = \bigvee E$, where E is a set of basic concave expressions. □

The class of systems of concave equations strictly subsumes the class of *systems of rational equations* and even the class of *systems of rational LP-equations* as defined by Gawlitza and Seidl [17, 22] (cf. [21]).

For this paper it is important to observe that every system of abstract semantic equations (cf. Section 2) is a system of concave equations: For every statement s, the expression $[\![s]\!]^\sharp_{j\cdot}(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ is a concave expression, since (1) the expression

$([\![s]\!]^\sharp_j.(\mathbf{x}_1,\ldots,\mathbf{x}_m))\sigma$ is a basic concave expression for all $\vee$-strategies σ, (i.e. $[\![s\sigma]\!]^\sharp_j.$ is monotone and concave) and (2) the expression $[\![s]\!]^\sharp_j.(\mathbf{x}_1,\ldots,\mathbf{x}_m)$ can be written as the expression $\bigvee_{\sigma\in\Sigma}([\![s]\!]^\sharp_j.(\mathbf{x}_1,\ldots,\mathbf{x}_m))\sigma$. Here, Σ denotes the set of all $\vee$-strategies. Hence, we can generalize the concept of $\vee$-strategies as follows:

Strategies A $\vee$-*strategy* σ for $\mathcal{E}$ is a function that maps every expression $\bigvee E$ occurring in $\mathcal{E}$ to one of the $e \in E$. We denote the set of all $\vee$-strategies for $\mathcal{E}$ by $\Sigma_\mathcal{E}$. We drop subscripts, whenever they are clear from the context. For $\sigma \in \Sigma$, the expression $e\sigma$ denotes the expression $\sigma(e)$. Finally, we set $\mathcal{E}(\sigma) := \{\mathbf{x} = e\sigma \mid \mathbf{x} = e \in \mathcal{E}\}$.

The Strategy Improvement Algorithm We briefly explain the strategy improvement algorithm (cf. [21, 22]). It iterates over $\vee$-strategies. It maintains a current $\vee$-strategy and a current *approximate* to the least solution. A so-called *strategy improvement operator* is used for determining a next, improved $\vee$-strategy. In our application, the strategy improvement operator is realized by a SAT modulo real linear arithmetic solver (cf. Section 4). Whether or not a $\vee$-strategy represents an *improvement* may depend on the current approximate. It can indeed be the case that a switch from one $\vee$-strategy to another $\vee$-strategy is only then *profitable*, when it is known, that the least solution is of a certain size. Hence, we talk about an *improvement* of a $\vee$-strategy w.r.t. an approximate:

Definition 4 (Improvements)**.** Let $\mathcal{E}$ be a system of monotone equations over a complete linear ordered set. Let $\sigma, \sigma' \in \Sigma$ be $\vee$-strategies for $\mathcal{E}$ and ρ be a pre-solution of $\mathcal{E}(\sigma)$. The $\vee$-strategy σ' is called *improvement of* σ *w.r.t.* ρ iff the following conditions are fulfilled: (1) If $\rho \notin \mathbf{Sol}(\mathcal{E})$, then $[\![\mathcal{E}(\sigma')]\!]\rho > \rho$. (2) For all $\bigvee$-expressions e occurring in $\mathcal{E}$ the following holds: If $\sigma'(e) \neq \sigma(e)$, then $[\![e\sigma']\!]\rho > [\![e\sigma]\!]\rho$. A function $P_\vee$ which assigns an improvement of σ w.r.t. ρ to every pair (σ, ρ), where σ is a $\vee$-strategy and ρ is a pre-solution of $\mathcal{E}(\sigma)$, is called $\vee$-*strategy improvement operator*. □

In many cases, there exist several, different improvements of a $\vee$-strategy σ w.r.t. a pre-solution ρ of $\mathcal{E}(\sigma)$. Accordingly, there exist several, different strategy improvement operators. One possibility for improving the current strategy is known as *all profitable switches* [4, 5]. Carried over to the case considered here, this means: For the improvement σ' of σ w.r.t. ρ we have: $[\![\mathcal{E}(\sigma')]\!]\rho = [\![\mathcal{E}]\!]\rho$, i.e., σ' represents the best local improvement of σ at ρ. We denote σ' by $P_\vee^{\mathsf{eager}}(\sigma, \rho)$ [17, 18, 19, 22].

Now we can formulate the strategy improvement algorithm for computing least solutions of systems of monotone equations over complete linear ordered sets. This algorithm is parameterized with a $\vee$-strategy improvement operator $P_\vee$. The input is a system $\mathcal{E}$ of monotone equations over a complete linear ordered set, a $\vee$-strategy σ_{init} for $\mathcal{E}$, and a pre-solution ρ_{init} of $\mathcal{E}(\sigma_{\text{init}})$. In order to compute the *least* and not some *arbitrary* solution, we additionally assume that $\rho_{\text{init}} \leq \mu[\![\mathcal{E}]\!]$ holds:

Algorithm 1 The Strategy Improvement Algorithm

Input : $\begin{cases} \text{- A system } \mathcal{E} \text{ of monotone equations over a complete linear ordered set} \\ \text{- A } \vee\text{-strategy } \sigma_{\text{init}} \text{ for } \mathcal{E} \\ \text{- A pre-solution } \rho_{\text{init}} \text{ of } \mathcal{E}(\sigma_{\text{init}}) \text{ with } \rho_{\text{init}} \leq \mu[\![\mathcal{E}]\!] \end{cases}$

$\sigma \leftarrow \sigma_{\text{init}}$; $\rho \leftarrow \rho_{\text{init}}$; **while** $(\rho \notin \mathbf{Sol}(\mathcal{E}))$ $\{\sigma \leftarrow P_\vee(\sigma, \rho)$; $\rho \leftarrow \mu_{\geq\rho}[\![\mathcal{E}(\sigma)]\!]; \}$ **return** ρ;

Lemma 13. *Let $\mathcal{E}$ be a system of monotone equations over a complete linear ordered set. For $i \in \mathbb{N}$, let ρ_i be the value of the program variable ρ and σ_i be the value of the program variable σ in the strategy improvement algorithm after the i-th evaluation of the loop-body. The following statements hold for all $i \in \mathbb{N}$:*

1. $\rho_i \leq \mu[\![\mathcal{E}]\!]$.
2. $\rho_i \in \mathbf{PreSol}(\mathcal{E}(\sigma_{i+1}))$.
3. *If $\rho_i < \mu[\![\mathcal{E}]\!]$, then $\rho_{i+1} > \rho_i$.*
4. *If $\rho_i = \mu[\![\mathcal{E}]\!]$, then $\rho_{i+1} = \rho_i$.* □

An immediate consequence of Lemma 13 is the following: Whenever the strategy improvement algorithm terminates, it computes the least solution $\mu[\![\mathcal{E}]\!]$ of $\mathcal{E}$.

At first we are interested in solving systems of concave equations with *finitely* many strategies and *finite* least solutions. We show that our strategy improvement algorithm terminates and thus returns the least solution in this case at the latest after considering all strategies. Further, we give an important characterization for $\mu_{\geq\rho}[\![\mathcal{E}(\sigma)]\!]$.

Feasibility In order to prove termination we define the following notion of feasibility:

Definition 5 (Feasibility ([21]))**.** Let $\mathcal{E}$ be a system of basic concave equations. A finite solution ρ of $\mathcal{E}$ is called *($\mathcal{E}$-)feasible* iff there exists $\mathbf{X}_1, \mathbf{X}_2 \subseteq \mathbf{X}$ and some $k \in \mathbb{N}$ such that the following statements hold:

1. $\mathbf{X}_1 \cup \mathbf{X}_2 = \mathbf{X}$, and $\mathbf{X}_1 \cap \mathbf{X}_2 = \emptyset$.
2. There exists some $\rho' \lhd \rho|_{\mathbf{X}_1}$ such that $\rho' \dot\cup \rho|_{\mathbf{X}_2}$ is a pre-solution of $\mathcal{E}$, and $\rho = [\![\mathcal{E}]\!]^k(\rho' \dot\cup \rho|_{\mathbf{X}_2})$.
3. There exists a $\rho' \lhd \rho|_{\mathbf{X}_2}$ such that $\rho' \lhd ([\![\mathcal{E}]\!]^k(\rho|_{\mathbf{X}_1} \dot\cup \rho'))|_{\mathbf{X}_2}$.

A finite pre-solution ρ of $\mathcal{E}$ is called *($\mathcal{E}$-)feasible* iff $\mu_{\geq\rho}[\![\mathcal{E}]\!]$ is a feasible finite solution of $\mathcal{E}$. A pre-solution $\rho \lhd \underline{\infty}$ is called feasible iff $e = -\infty$ for all $\mathbf{x} = e \in \mathcal{E}$ with $[\![e]\!]\rho = -\infty$, and $\rho|_{\mathbf{X}'}$ is a feasible finite pre-solution of $\{\mathbf{x} = e \in \mathcal{E} \mid \mathbf{x} \in \mathbf{X}'\}$, where $\mathbf{X}' := \{\mathbf{x} \mid \mathbf{x} = e \in \mathcal{E}, [\![e]\!]\rho > -\infty\}$.

A system $\mathcal{E}$ of basic concave equations is called *feasible* iff there exists a feasible solution ρ of $\mathcal{E}$. □

The following lemmas ensure that our strategy improvement algorithm stays in the feasible area, whenever it is started in the feasible area.

Lemma 14 ([21])**.** *Let $\mathcal{E}$ be a system of basic concave equations and ρ be a feasible pre-solution of $\mathcal{E}$. Every pre-solution ρ' of $\mathcal{E}$ with $\rho \leq \rho' \leq \mu_{\geq\rho}[\![\mathcal{E}]\!]$ is feasible.* □

Lemma 15 ([21])**.** *Let $\mathcal{E}$ be a system of concave equations, σ be a $\vee$-strategy for $\mathcal{E}$, ρ be a feasible solution of $\mathcal{E}(\sigma)$, and σ' be an improvement of σ w.r.t. ρ. Then ρ is a feasible pre-solution of $\mathcal{E}(\sigma')$.* □

In order to start in the feasible area, we simply start the strategy improvement algorithm with the system $\mathcal{E} \vee -\infty := \{\mathbf{x} = e \vee -\infty \mid \mathbf{x} = e \in \mathcal{E}\}$, a $\vee$-strategy σ_{init} for $\mathcal{E} \vee -\infty$ such that $(\mathcal{E} \vee -\infty)(\sigma_{\text{init}}) = \{\mathbf{x} = -\infty \mid \mathbf{x} = e \in \mathcal{E}\}$, and the feasible pre-solution $\underline{-\infty}$ of $(\mathcal{E} \vee -\infty)(\sigma_{\text{init}})$.

It remains to determine $\mu_{\geq\rho}[\![\mathcal{E}]\!]$. Because of Lemma 14 and Lemma 15, we are allowed to assume that ρ is a feasible pre-solution of the system $\mathcal{E}$ of basic concave equations. This is important in our strategy improvement algorithm. The following lemma in particular states that we have to compute the greatest finite pre-solution.

Lemma 16 ([21])**.** *Let $\mathcal{E}$ be a feasible system of basic concave equations with $e \neq -\infty$ for all $\mathbf{x} = e \in \mathcal{E}$. There exists a greatest finite pre-solution ρ^* of $\mathcal{E}$ and ρ^* is the only feasible solution of $\mathcal{E}$. If ρ is a finite pre-solution of $\mathcal{E}$, then $\rho^* = \mu_{\geq\rho}[\![\mathcal{E}]\!]$.* □

Termination Lemma 16 implies that our strategy improvement algorithm has to consider each $\vee$-strategy at most once. Thus, we have shown the following theorem:

Theorem 17. *Let $\mathcal{E}$ be a system of concave equations with $\mu[\![\mathcal{E}]\!] \lhd \underline{\infty}$. Assume that we can compute the greatest finite pre-solution ρ_σ of each $\mathcal{E}(\sigma)$, if $\mathcal{E}(\sigma)$ is feasible. Our strategy improvement algorithm computes $\mu[\![\mathcal{E}]\!]$ and performs at most $|\Sigma| + |\mathbf{X}|$ strategy improvement steps. The algorithm in particular terminates, whenever Σ is finite.* □

6 Computing Greatest Finite Pre-Solutions

For all systems $\mathcal{E}$ of abstract semantic equations (see Section 2) and all $\vee$-strategies σ, $\mathcal{E}(\sigma)$ is a system of abstract semantic equations, where each right-hand side is of the form $[\![s]\!]^\sharp_{j\cdot}(\mathbf{x}_1, \ldots, \mathbf{x}_m)$, where s is a sequential statement and $\mathbf{x}_1, \ldots, \mathbf{x}_m$ are variables. We call such a system of abstract semantic equations a system of *basic* abstract semantic equations. It remains to explain how we can compute the greatest finite solution of such a system — provided that it exists.

Let $\mathcal{E}$ be a system of basic abstract semantic equations with a greatest finite pre-solution ρ^*. We can compute ρ^* through linear programming as follows:

We assume w.l.o.g. that every sequential statement s that occurs in the right-hand sides of $\mathcal{E}$ is of the form $Ax \leq b; x := A'x + b'$, where $A \in \mathbb{R}^{k\times n}, b \in \mathbb{R}^k, A' \in \mathbb{R}^{n\times n}, b' \in \mathbb{R}^n$. This can be done w.l.o.g., since every sequential statement can be rewritten into this form in polynomial time. We define the system $\mathcal{C}$ of linear inequalities to be the smallest set that fulfills the following properties: For each equation

$$\mathbf{x} = [\![Ax \leq b; x := A'x + b']\!]^\sharp_{j\cdot}(\mathbf{x}_1, \ldots, \mathbf{x}_m),$$

the system $\mathcal{C}$ contains the following constraints:

$$\mathbf{x} \leq T_{j\cdot}A'(\mathbf{y}_1, \ldots, \mathbf{y}_n)^\top + T_{j\cdot}b' \qquad A_{i\cdot}(\mathbf{y}_1, \ldots, \mathbf{y}_n)^\top \leq b_i \text{ for all } i = 1, \ldots, k$$
$$T_{i\cdot}(\mathbf{y}_1, \ldots, \mathbf{y}_n)^\top \leq \mathbf{x}_i \text{ for all } i = 1, \ldots, m$$

Here, $\mathbf{y}_1, \ldots, \mathbf{y}_n$ are fresh variables. Then $\rho^*(\mathbf{x}) = \sup\{\rho(\mathbf{x}) \mid \rho \in \mathbf{Sol}(\mathcal{C})\}$. Thus ρ^* can be determined by solving $|\mathbf{X}_\mathcal{E}|$ linear programming problems each of which can be constructed in linear time. We can do even better by determining an optimal

solution of the linear programming problem $\sup\left\{\sum_{\mathbf{x}\in\mathbf{X}_\mathcal{E}}\rho(\mathbf{x}) \mid \rho\in\mathbf{Sol}(\mathcal{C})\right\}$. Then the optimal values for the variables $\mathbf{x}\in\mathbf{X}_\mathcal{E}$ determine ρ^* (cf. Gawlitza and Seidl [17, 22]). Summarizing we have:

Lemma 18. *Let $\mathcal{E}$ be a system of basic abstract semantic equations with a greatest finite pre-solution ρ^*. Then ρ^* can be computed by solving a linear programming problem that can be constructed in linear time.* □

Example 19. We again use the definitions of Example 7. Consider the system $\mathcal{E}$ of basic abstract semantic equations that consists of the equations

$$\mathbf{x}_{1,1} = [\![s';s_2]\!]^\sharp_{1.}(\mathbf{x}_{1,1},\mathbf{x}_{1,2}) \qquad \mathbf{x}_{1,2} = [\![s';s_1]\!]^\sharp_{2.}(\mathbf{x}_{1,1},\mathbf{x}_{1,2}),$$

where $s' := x_1 \leq 1000; x_2 := -x_1$, $s_1 := x_2 \leq -1; x_1 := -2x_1$, and $s_2 := -x_2 \leq 0; x_1 := -x_1 + 1$. Our goal is to compute the greatest finite pre-solution ρ^* of $\mathcal{E}$. Firstly, we note that $[\![s';s_2]\!] = [\![x_1 \leq 0; (x_1,x_2) := (-x_1+1,-x_1)]\!]$ and $[\![s';s_1]\!] = [\![(x_1,-x_1) \leq (1000,-1); (x_1,x_2) := (-2x_1,-x_1)]\!]$ hold. Accordingly, we have to find an optimal solution for the following linear programming problem:

maximize $\mathbf{x}_{1,1} + \mathbf{x}_{1,2}$

$\mathbf{x}_{1,1} \leq -\mathbf{y}_1 + 1$	$\mathbf{x}_{1,2} \leq 2\mathbf{y}'_1$	$\mathbf{y}_1 \leq 0$	$\mathbf{y}'_1 \leq 1000$	$\mathbf{y}_1 \leq \mathbf{x}_{1,1}$
$-\mathbf{y}'_1 \leq -1$	$-\mathbf{y}_1 \leq \mathbf{x}_{1,2}$	$\mathbf{y}'_1 \leq \mathbf{x}_{1,1}$	$-\mathbf{y}'_1 \leq \mathbf{x}_{1,2}$	

An optimal solution is $\mathbf{x}_{1,1} = 2001$, $\mathbf{x}_{1,2} = 2000$, $\mathbf{y}_1 = -2000$, and $\mathbf{y}'_1 = 1000$. Thus $\rho^* = \{\mathbf{x}_{1,1} \mapsto 2001,\ \mathbf{x}_{1,2} \mapsto 2000\}$ is the greatest finite pre-solution of $\mathcal{E}$. □

Summarizing, we have shown our main theorem:

Theorem 20. *Let $\mathcal{E}$ be a system of abstract semantic equations with $\mu[\![\mathcal{E}]\!] \lhd \underline{\infty}$. Our strategy improvement algorithm computes $\mu[\![\mathcal{E}]\!]$ and performs at most $|\Sigma| + |\mathbf{X}|$ strategy improvement steps. For each strategy improvement step, we have to do the following:*

1. *Find models for $|\mathbf{X}|$ SAT modulo real linear arithmetic formulas, each of which can be constructed in linear time.*
2. *Solve a linear programming problem which can be constructed in linear time.*

Proof. The statement follows from Lemmas 14, 15, 16, 18 and Theorem 17. □ □

Our techniques can be extended straightforwardly in order to get rid of the pre-condition $\mu[\![\mathcal{E}]\!] \lhd \underline{\infty}$. However, for simplicity we eschew these technicalities in the present article.

7 An Upper Bound on the Complexity

In Section 3, we have provided a lower bound on the complexity of computing abstract semantics of affine programs w.r.t. the template linear domains. In this section we show that the corresponding decision problem is not only Π^p_2-hard, but in fact Π^p_2-complete:

Theorem 21. *The problem of deciding, whether or not, for a given affine program G, a given template constraint matrix T, and a given program point v, $V^\sharp[v] > \underline{-\infty}$ holds, is in Π^p_2.*

Proof. (Sketch) We have to show that the problem of deciding, whether or not, for a given affine program G, a given template constraint matrix T, a given program point v, and a given $i \in \{1, \ldots, m\}$, $(V^\sharp[v])_{i\cdot} = -\infty$ holds, is in $\mathsf{co}{-}\Pi_2^p = \Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$. In polynomial time we can guess a $\vee$-strategy σ for $\mathcal{E}' := \mathcal{E}(G)$ and compute the *least feasible solution* ρ of $\mathcal{E}'(\sigma)$ (see Gawlitza and Seidl [17]). Because of Lemma 4, we can use a NP oracle to determine whether or not there exists an improvement of the strategy σ w.r.t. ρ. If this is not the case, we know that $\rho \geq \mu[\![\mathcal{E}']\!]$ holds. Therefore, by Lemma 6, we have $\rho(\mathbf{x}_{v,i}) \geq (V^\sharp[v])_{i\cdot}$. Thus we can accept, whenever $\rho(\mathbf{x}_{v,i}) = -\infty$ holds. □ □

Finally, we give an example where our strategy improvement algorithm performs exponentially many strategy improvement steps. It is similar to the program in the proof of Theorem 8. For all $n \in \mathbb{N}$, we consider the program $G_n = (N, E, \mathbf{st})$, where $N = \{\mathbf{st}, 1\}$, $E = \{(\mathbf{st}, x_1 := 0; y_1 := 1; y_2 := 2y_1; \ldots; y_n := 2y_{n-1}, 1), (1, s, 1)\}$, and

$$\begin{aligned} s \;=\; & x_2 := x_1; (x_2 \geq y_n; x_2 := x_2 - y_n \mid x_2 \leq y_n - 1); \cdots \\ & (x_2 \geq y_1; x_2 := x_2 - y_1 \mid x_2 \leq y_1 - 1);\; x_1 := x_1 + 1. \end{aligned}$$

It is sufficient to use a template constraint matrix that corresponds to the interval domain. It is remarkable that the strategy iteration does not depend on the strategy improvement operator in use. At any time there is exactly one possible improvement until the least solution is reached. All strategies for the statement s will be encountered. Thus, the strategy improvement algorithm performs 2^n strategy improvement steps. Since the size of G_n is $\Theta(n)$, exponentially many strategy improvement steps are performed.

8 Conclusion

We presented an extension of the strategy improvement algorithm of Gawlitza and Seidl [17, 18, 21] which enables us to use a SAT modulo real linear arithmetic solver for determining improvements of strategies w.r.t. current approximates. Due to this extension, we are able to compute abstract semantics of affine programs w.r.t. the template linear constraint domains of Sankaranarayanan et al. [42], where we abstract sequences of if-then-else statements without loops en bloc. This gives us additional precision. Additionally, We provided one of the few "hard" complexity results regarding precise abstract interpretation.

It remains to practically evaluate the presented approach and to compare it systematically with other approaches. Besides this, starting from the present work, there are several directions to explore. One can for instance try to apply the same ideas for non-linear templates [21], or to use linearization techniques [35].

References

[1] A. Adjé, S. Gaubert, and E. Goubault. Computing the smallest fixed point of nonexpansive mappings arising in game theory and static analysis of programs. *ArXiv e-prints*, June 2008. 0806.1160v2.

[2] Assalé Adjé, Stephane Gaubert, and Eric Goubault. Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. In Andrew D. Gordon, editor, *ESOP*, volume 6012 of *LNCS*, pages 23–42. Springer, 2010. ISBN 978-3-642-11956-9.

[3] Thomas Ball and Robert B. Jones, editors. *Computer Aided Verification (CAV)*, volume 4144 of *LNCS*, 2006. Springer. ISBN 3-540-37406-X.

[4] H. Björklund, S. Sandberg, and S. Vorobyov. Optimization on completely unimodal hypercubes. Technical report 2002-18, Uppsala University, 2002.

[5] Henrik Bjorklund, Sven Sandberg, and Sergei Vorobyov. Complexity of Model Checking by Iterative Improvement: the Pseudo-Boolean Framework . In *Proc. 5th Int. Andrei Ershov Memorial Conf. Perspectives of System Informatics*, pages 381–394. LNCS 2890, Springer, 2003. doi: 10.1007/978-3-540-39866-0_38.

[6] Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. A static analyzer for large safety-critical software. In *Programming Language Design and Implementation (PLDI)*, pages 196–207. ACM, 2003. ISBN 1-58113-662-5. doi: 10.1145/781131.781153.

[7] Jean Cochet-Terrasson, Stéphane Gaubert, and Jeremy Gunawardena. A Constructive Fixed Point Theorem for Min-Max Functions. *Dynamics and Stability of Systems*, 14(4):407–433, 1999.

[8] Michael A. Colón, Sriram Sankaranarayanan, and Henny Sipma. Linear invariant generation using non-linear constraint solving. In *Computer Aided Verification (CAV)*, number 2725 in LNCS, pages 420–433. Springer, 2003. ISBN 3-540-40524-0. doi: 10.1007/b11831.

[9] Alexandru Costan, Stephane Gaubert, Eric Goubault, Matthieu Martel, and Sylvie Putot. A Policy Iteration Algorithm for Computing Fixed Points in Static Analysis of Programs. In *Computer Aided Verification, 17th Int. Conf. (CAV)*, pages 462–475. LNCS 3576, Springer Verlag, 2005. ISBN 3-540-27231-3. doi: 10.1007/11513988_46.

[10] Patrick Cousot. Proving program invariance and termination by parametric abstraction, Lagrangian relaxation and semidefinite programming. In Radhia Cousot, editor, *Verification, Model Checking and Abstract Interpretation (VMCAI)*, number 3385 in LNCS, pages 1–24. Springer, 2005. ISBN 3-540-24297-X. doi: 10.1007/b105073.

[11] Patrick Cousot and Radhia Cousot. Static Determination of Dynamic Properties of Programs. In *Second Int. Symp. on Programming*, pages 106–130. Dunod, Paris, France, 1976.

[12] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977. doi: 10.1145/512950.512973.

[13] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL*, pages 84–96, 1978. doi: 10.1145/512760.512770.

[14] Bruno Dutertre and Leonardo de Moura. The Yices SMT solver. Tool paper at `http://yices.csl.sri.com/tool-paper.pdf`, August 2006.

[15] Bruno Dutertre and Leonardo Mendonça de Moura. A fast linear-arithmetic solver for DPLL(T). In Ball and Jones [3], pages 81–94. ISBN 3-540-37406-X. doi: 10.1007/11817963_11.

[16] Stephane Gaubert, Eric Goubault, Ankur Taly, and Sarah Zennou. Static analysis by policy iteration on relational domains. In Nicola [38], pages 237–252. ISBN 978-3-540-71314-2.

[17] Thomas Gawlitza and Helmut Seidl. Precise relational invariants through strategy iteration. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL*, volume 4646 of *LNCS*, pages 23–40. Springer, 2007. ISBN 978-3-540-74914-1.

[18] Thomas Gawlitza and Helmut Seidl. Precise fixpoint computation through strategy iteration. In Nicola [38], pages 300–315. ISBN 978-3-540-71314-2.

[19] Thomas Gawlitza and Helmut Seidl. Precise interval analysis vs. parity games. In Jorge Cuéllar, T. S. E. Maibaum, and Kaisa Sere, editors, *FM*, volume 5014 of *LNCS*, pages 342–357. Springer, 2008. ISBN 978-3-540-68235-6.

[20] Thomas Gawlitza, Jérôme Leroux, Jan Reineke, Helmut Seidl, Grégoire Sutre, and Reinhard Wilhelm. Polynomial precise interval analysis revisited. In Susanne Albers, Helmut Alt, and Stefan Näher, editors, *Efficient Algorithms*, volume 5760 of *LNCS*, pages 422–437. Springer, 2009. ISBN 978-3-642-03455-8.

[21] Thomas Martin Gawlitza and Helmut Seidl. Computing relaxed abstract semantics w.r.t. quadratic zones precisely. In *SAS*, volume 6337 of *LNCS*, pages 271–286. Springer, 2010. ISBN 3-642-15768-8. doi: 10.1007/978-3-642-15769-1_17.

[22] Thomas Martin Gawlitza and Helmut Seidl. Solving systems of rational equations through strategy iteration. Technical report, TUM, 2009.

[23] Laure Gonnord. *Accélération abstraite pour l'amélioration de la précision en analyse des relations linéaires*. PhD thesis, Université Joseph Fourier, October 2007. URL `http://tel.archives-ouvertes.fr/tel-00196899/en/`.

[24] Laure Gonnord and Nicolas Halbwachs. Combining widening and acceleration in linear relation analysis. In Kwangkeun Yi, editor, *SAS*, volume 4134 of *LNCS*, pages 144–160. Springer, 2006. ISBN 3-540-37756-5.

[25] Denis Gopan and Thomas W. Reps. Lookahead widening. In Ball and Jones [3], pages 452–466. ISBN 3-540-37406-X. doi: 10.1007/11817963_41.

[26] H. G. Rice. Classes of recursively enumerable sets and their decision problems. In *Transactions of the American Mathematical Society*, volume 74, pages 358–366. AMS, 1953.

[27] Nicolas Halbwachs. Delay analysis in synchronous programs. In Costas Courcoubetis, editor, *Computer Aided Verification (CAV)*, volume 697 of *LNCS*, pages 333–346. Springer, 1993. ISBN 3-540-56922-7. doi: 10.1007/3-540-56922-7_28.

[28] A.J. Hoffman and R.M. Karp. On Nonterminating Stochastic Games. *Management Sci.*, 12:359–370, 1966.

[29] R. Howard. *Dynamic Programming and Markov Processes*. Wiley, NY, 1960.

[30] Jeremy Leconte, Stephane Le Roux, Leo Liberti, and Fabrizio Marinelli. Code verification by static analysis: a mathematical programming approach. Technical report, LIX, Ecole Polytechnique, Palaiseau, August 2009.

[31] Jérôme Leroux and Grégoire Sutre. Accelerated data-flow analysis. In *Static Analysis (SAS)*, volume 4634 of *LNCS*, pages 184–199. Springer, 2007. doi: 10.1007/s10009-008-0064-3.

[32] Nimrod Megiddo. On the Complexity of Linear Programming. In T. Bewley, editor, *Advances in Economic Theory: 5th World Congress*, pages 225–268. Cambridge University Press, 1987.

[33] Antoine Miné. A new numerical abstract domain based on difference-bound matrices. In Olivier Danvy and Andrzej Filinski, editors, *PADO*, volume 2053 of *LNCS*, pages 155–172. Springer, 2001. ISBN 3-540-42068-1.

[34] Antoine Miné. The octagon abstract domain. In *WCRE*, pages 310–, 2001.

[35] Antoine Miné. *Domaines numériques abstraits faiblement relationnels*. PhD thesis, École polytechnique, 2004.

[36] David Monniaux. A quantifier elimination algorithm for linear real arithmetic. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *LPAR*, volume 5330 of *LNCS*, pages 243–257. Springer, 2008. ISBN 978-3-540-89438-4.

[37] David Monniaux. Automatic modular abstractions for linear constraints. In Zhong Shao and Benjamin C. Pierce, editors, *POPL*, pages 140–151. ACM, 2009. ISBN 978-1-60558-379-2.

[38] Rocco De Nicola, editor. *Programming Languages and Systems, ESOP 2007, Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4421 of *LNCS*, 2007. Springer. ISBN 978-3-540-71314-2.

[39] Anuj Puri. *Theory of Hybrid and Discrete Systems*. PhD thesis, University of California, Berkeley, 1995.

[40] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.

[41] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. Constraint-based linear-relations analysis. In *Static Analysis (SAS)*, number 3148 in LNCS, pages 53–68. Springer, 2004. doi: 10.1007/b99688.

[42] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Scalable analysis of linear systems using mathematical programming. In Radhia Cousot, editor, *VMCAI*, volume 3385 of *LNCS*, pages 25–41. Springer, 2005. ISBN 3-540-24297-X.

[43] Alexandeer Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986. ISBN 0-471-90854-1.

[44] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, October 1976. doi: 10.1016/0304-3975(76)90061-X.

[45] Jens Vöge and Marcin Jurdziński. A Discrete Strategy Improvement Algorithm for Solving Parity Games. In *Computer Aided Verification, 12th Int. Conf. (CAV)*, pages 202–215. LNCS 1855, Springer, 2000.

[46] Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):23–33, 1976. doi: 10.1016/0304-3975(76)90062-1.