Scalable Analysis of Linear Systems using Mathematical Programming

Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna *

Computer Science Department
Stanford University
Stanford, CA 94305-9045
{srirams,sipma,zm}@theory.stanford.edu

Abstract. We present a method for generating linear invariants for large systems. The method performs forward propagation in an abstract domain consisting of arbitrary polyhedra of a predefined fixed shape. The basic operations on the domain like abstraction, intersection, join and inclusion tests are all posed as linear optimization queries, which can be solved efficiently by existing LP solvers. The number and dimensionality of the LP queries are polynomial in the program dimensionality, size and the number of target invariants. The method generalizes similar analyses in the interval, octagon, and octahedra domains, without resorting to polyhedral manipulations. We demonstrate the performance of our method on some benchmark programs.

1 Introduction

Static analysis is one of the central challenges in computer science, and increasingly, in other disciplines such as computational biology. Static analysis seeks to discover invariant relationships between the variables of a system that hold on every execution of the system. In computer science, knowledge of these relationships is invaluable for verification and optimization of systems; in computational biology this knowledge may lead to better understanding of the system's dynamics.

Linear invariant generation, the discovery of linear relationships between variables, has a long history, starting with Karr [9], and cast in the general framework of abstract interpretation by Cousot and Cousot [6]. The most general form of linear invariant generation is *polyhedral analysis*. The analysis is performed in the abstract domain of all the linear inequalities over all the system variables [7]. Although impressive results have been achieved in this domain, its applicability is severely limited by its worst-case exponential time and space complexity. This has led to the investigation of more restricted domains which seek to trade off

* This research was supported in part by NSF grants CCR-01-21403, CCR-02-20134 and CCR-02-09237, by ARO grant DAAD19-01-1-0723, by ARPA/AF contracts F33615-00-C-1693 and F33615-99-C-3014, by NAVY/ONR contract N00014-03-1-0939, and by the Siebel Graduate Fellowship.

some precision against tractability. The *interval domain* consists of inequalities of the form $a \leq x_i \leq b$. This was first studied by Cousot and Cousot [5]. Miné et al. consider the abstract domain of inequalities of the form $x_i - x_j \leq c$, known as *Difference-Bound Matrices* [12], and more generally, inequalities of the form $\pm x_i \pm x_j \leq c$, known as the *Octagon abstract domain* [13]. The domain has been applied to large programs with impressive results [3]. More recently, Clarisó et al. generalized octagons to *octahedra*, inequalities of the form $a_1x_1 + \ldots a_nx_n \leq c$, where each a_i is either ± 1 or 0, and applied it to the verification of timing delays in asynchronous circuits [4].

In this paper, we show that an efficient forward propagation-based analysis can be performed in an abstract domain that lies between the interval domain of Cousot&Cousot [5], and the general polyhedra [7]. Our proposed domain can contain any inequality of the form $a_1x_1 + \ldots a_nx_n + c \geq 0$. It requires the coefficients $a_1, \ldots, a_n$ for all inequalities in the abstract domain to be fixed in advance, and thus is less general than polyhedra. Since $a_1, \ldots, a_n$ can be user specified, our domain neatly subsumes the body of work described above. We show that all such analyses can be conducted in worst-case polynomial time in the program size and the domain size.

The rest of the paper is organized as follows: Section 2 reviews the basic theory of polyhedra, linear programming, system models and abstract interpretation. In Section 3, we describe our abstract domain, followed by the analysis algorithm and strategies on abstract domain construction. Section 4 discusses the complexity of our algorithm, and presents the results of applying it to several benchmark programs.

2 Preliminaries

We recall some standard results on polyhedra, followed by a brief description of system models and abstract interpretation. Throughout the paper, $x_1, \ldots, x_n$ denote real-valued variables, a, b with subscripts denote constant reals and c, d denote unknown coefficients. Similarly A, B denote real matrices, while A_i represents the ith row of the matrix A. We let $\boldsymbol{a}, \ldots, \boldsymbol{z}$ denote vectors. A vector is also a $n \times 1$ column-matrix for $n \geq 0$. The relation $\boldsymbol{a} \leq \boldsymbol{b}$ is used to denote $a_i \leq b_i$ for all $i = 1 \ldots n$.

2.1 Polyhedra

Definition 1 (Linear Assertions) A *linear inequality* is an expression of the form $a_1x_1 + \cdots + a_nx_n + b \bowtie 0$, and $\bowtie \in \{\geq, =\}$. A *linear assertion* is a finite conjunction of linear inequalities. The assertion

$$\varphi : \begin{bmatrix} a_{11}x_1 & + \ldots + & a_{1n}x_n & + & b_1 & \geq 0 & \wedge \\ \vdots & \ldots & \vdots & & \vdots & & \vdots \\ a_{m1}x_1 & + \ldots + & a_{mn}x_n & + & b_m & \geq 0 & \end{bmatrix}$$

can concisely be written in matrix form as $A\boldsymbol{x}+\boldsymbol{b} \geq \mathbf{0}$, where $\boldsymbol{x}$ and $\boldsymbol{b}$ are n and m-dimensional vectors, respectively. The set of points in $\mathcal{R}^n$ satisfying a linear assertion is called a *polyhedron*. Polyhedra can be represented implicitly by a linear assertion, also known as the *constraint representation*, or explicitly by a set of vertices and rays, also known as the *generator representation* [15]. In this paper we assume that linear assertions do not contain any strict inequalities.

The linear consequences of a linear assertion φ, that is, the linear inequalities that are implied by φ, can be deduced using Farkas Lemma [15]:

Theorem 1 (Farkas Lemma). *Consider the linear assertion*

$$\varphi : \; A\boldsymbol{x} + \boldsymbol{b} \geq 0$$

over real-valued variables $\boldsymbol{x}$. *If* φ *is satisfiable, then it implies the linear inequality* $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + d \geq 0$ *iff there exists* $\boldsymbol{\lambda} \geq 0$ *such that*

$$A^{\mathrm{T}}\boldsymbol{\lambda} = \boldsymbol{c} \text{ and } \boldsymbol{b}^{\mathrm{T}}\boldsymbol{\lambda} \leq d.$$

Furthermore, φ *is unsatisfiable iff there exists* $\boldsymbol{\lambda} \geq 0$ *such that*

$$A^{\mathrm{T}}\boldsymbol{\lambda} = \mathbf{0} \text{ and } \boldsymbol{b}^{\mathrm{T}}\boldsymbol{\lambda} \leq -1.$$

The main engine behind our analysis is a Linear Programming (LP) solver. We shall describe the theory of linear programming briefly.

Definition 2 (Linear Programming) An instance of the *linear programming (LP) problem* consists of a linear assertion φ and a linear expression $f : \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}$, called the *objective function*. The goal is to determine the solution of φ for which f is minimal. An LP problem can have one of three results: (1) an optimal solution; (2) φ has solutions, but none is optimal with respect to f (f is unbounded in φ); (3) φ has no solutions.

In principle, an LP problem can be solved by computing the generators of the polyhedron corresponding to φ. If the polyhedron is empty, i.e., it has no generators, then there are no solutions. If the polyhedron has a ray along which the objective function f decreases, then f is unbounded. Also, it has been demonstrated that an optimal solution (if it exists) is realized at a vertex of the polyhedron. The optimal solution can be found by evaluating f at each of the vertices. Enumerating all the generators is very inefficient because the number of generators is worst-case exponential in the number of constraints. The popular SIMPLEX algorithm, although worst-case exponential in theory, is very fast over most problems in practice. Our method scales by taking advantage of this fact. Interior point methods like the *Karmarkar's* algorithm can solve linear programs in polynomial time. In practice, we shall use SIMPLEX for our linear programming needs because of its free availability and its numerical stability.

2.2 Transition Systems and Invariants

As computational model, we use transition systems [11]. For ease of exposition we assume that the transition systems are linear, as defined below. Any transition can be linearized by omitting all nonlinear constructs from the initial condition and transition relations.

Definition 3 (Linear Transition Systems) A linear transition system $S : \langle L, \mathcal{T}, \ell_0, \Theta \rangle$ over a set of variables V consists of

- L: a set of locations;
- $\mathcal{T}$: a set of transitions, where each transition $\tau : \langle \ell_i, \ell_j, \rho_\tau \rangle$ consists of a pre-location ℓ_i, a post-location ℓ_j, and a transition relation ρ_τ that is a linear assertion over $V \cup V'$, where V denotes the values of the variables in the current state, and V' their values in the next state;
- $\ell_0 \in L$: the initial location;
- Θ: a linear assertion over V specifying the initial condition.

A *run* of a linear transition system is a sequence of pairs $(l_0, s_0), (l_1, s_1), (l_2, s_2), \ldots$ with $l_i \in L$ and s_i a valuation of V, also called a *state*, such that

- Initiation: $l_0 = \ell_0$, and $s_0 \models \Theta$
- Consecution: for all $i \geq 0$ there exists a transition $\tau : \langle p, q, \rho_\tau \rangle$ such that $l_i = p$, $l_{i+1} = q$, and $(s_i, s_{i+1}) \models \rho_\tau$.

A state s is *reachable* at location l if (l, s) appears in some run.

Henceforth, we shall assume that transitions are limited to guarded assignments of the form $\xi \wedge \boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{b}$, where the *guard* ξ is a linear assertion over V, and A, $\boldsymbol{b}$ are matrices. This form is common in transition systems derived from programs. However, the results easily extend to the general case too.

Example 1. Following is a transition system over $V = \{x, y\}$ with one location and two transitions that update the variables x and y atomically:

$$\begin{array}{lcl} L & : & \{l_0\} \\ \mathcal{T} & : & \{\tau_1, \tau_2\} \quad \text{with} \quad \begin{cases} \tau_1 : \left\langle l_0, l_0, \left[x' = x + 2y \ \wedge \ y' = 1 - y\right] \right\rangle \\ \tau_2 : \left\langle l_0, l_0, \left[x' = x + 1 \ \wedge \ y' = y + 2\right] \right\rangle \end{cases} \\ \ell_0 & : & l_0 \\ \Theta & : & (x = 0 \ \wedge \ y = 0) \end{array}$$

A given linear assertion ψ is a *linear invariant* of a linear transition system (LTS) at a location ℓ iff it is satisfied by every state reaching ℓ. An *assertion map* maps each location of a LTS to a linear assertion. An assertion map η is an invariant map if $\eta(\ell)$ is an invariant at ℓ, for each $\ell \in L$. In order to prove a given assertion map invariant, we use the theory of inductive assertions [11].

Definition 4 (Inductive Assertion Maps) An assertion map η is inductive iff it satisfies the following conditions:

Initiation: $\Theta \models \eta(\ell_0)$,
Consecution: For each transition $\tau : \langle \ell_i, \ell_j, \rho_\tau \rangle$, $\eta(\ell_i) \ \wedge \ \rho_\tau \ \models \eta(\ell_j)'$.

It can be proven by mathematical induction that any inductive assertion map is also an invariant map. It is well known that the converse need not be true in general. The standard technique for proving an assertion invariant is to find an inductive assertion that strengthens it. For example, the assertion $x + y \geq 0$ is an inductive assertion for the LTS in Example 1.

2.3 Propagation-based Analysis

Forward propagation consists of a symbolic simulation of the program to compute an assertion representing the reachable state space. Starting with the initial condition, the assertion is iteratively weakened by adding states that are reachable in one step, as computed by the post operator, $post(\tau, \varphi) : \ \exists V_0 \ . \ (\varphi(V_0) \wedge \rho_\tau(V_0, V))$ until no more states can be added. This procedure can be described as the computation of a fixed point of the second order function (predicate transformer)

$$\mathfrak{F}(X) \ = \ \Theta \vee X \vee \bigvee_{\tau \in \mathcal{T}} post(\tau, X)$$

starting from $\mathfrak{F}(false)$. The least fixed point describes exactly the reachable state space.

This approach has two problems: (1) the sequence $\mathfrak{F}(false), \mathfrak{F}^2(false), \ldots$ may not converge in a finite number of steps, and (2) we may not be able to detect convergence, because the inclusion $\mathfrak{F}^{n+1}(false) \subseteq \mathfrak{F}^n(false)$ may be undecidable. These problems were addressed by the *abstract interpretation framework* formalized by Cousot and Cousot [6], and specialized for linear relations by Cousot and Halbwachs [7].

The abstract interpretation framework performs the forward propagation in a simpler, abstract domain, in which the detection of convergence is decidable. Also the resulting fixed point, when translated back, is guaranteed to be a fixed point (though not necessarily a least fixed point) of the concrete predicate transformer. The problem of finite convergence was addressed by the introduction of a *widening* operator that guarantees termination in a finite number of steps.

The application of abstract interpretation requires the definition of an abstract domain Σ_A, equipped with a partial order $\leq_A$, an abstraction function $\alpha : \ 2^{\Sigma} \mapsto \Sigma_A$ that maps sets of states into elements in the abstract domain, and a concretization function $\gamma : \ \Sigma_A \mapsto 2^{\Sigma}$. The functions α and γ must form a *Galois connection*, that is they must satisfy $\alpha(S) \leq_A a$ iff $S \subseteq \gamma(a)$ for all $S \subseteq \Sigma$ and $a \in \Sigma_A$.

Forward propagation can now be performed in the abstract domain by computing the fixed point of

$$\mathfrak{F}_A(X) \ = \ \Theta_A \sqcup X \sqcup \bigsqcup_{\tau \in \mathcal{T}} post_A(\tau, X) \ .$$

If the operations $\sqcup$ and $post_A$ satisfy $\gamma(a_1) \vee \gamma(a_2) \subseteq \gamma(a_1 \sqcup a_2)$, and $post(\tau, \gamma(a)) \subseteq \gamma(post_A(a))$, and the abstract element Θ_A satisfies $\Theta \subseteq \gamma(\Theta_A)$, then $\gamma(lfp(\mathfrak{F}_A))$ is guaranteed to be a fixed point of $\mathfrak{F}$.

Polyhedra are a very popular abstract domain. Checking for inclusion is decidable, and effective widening operators have been designed. However, manipulating large polyhedra remains computationally expensive and hence, this analysis does not scale very well in practice.

Note. Throughout the rest of the paper, instead of the traditional $\subseteq$ relation, we shall use the models relation ($\models$) between formulas as the order on the concrete domain.

3 Invariant Generation Algorithm

3.1 Abstract Domain

Our abstract domain consists of polyhedra of a fixed shape for a given set of variables $\boldsymbol{x}$ of cardinality n. The shape is fixed by an $m \times n$ template constraint matrix (TCM) T. If T is nonempty, i.e, $m > 0$, the abstract domain Σ_T contains m-dimensional vectors $\boldsymbol{c}$. Each entry c_i may be real-valued, or a special-valued entry drawn from the set $\{\infty, -\infty\}$. A vector $\boldsymbol{c}$ in the abstract domain Σ_T represents the set of states described by the set of constraints $T\boldsymbol{x} + \boldsymbol{c} \geq \mathbf{0}$. If the TCM T is empty, that is $m = 0$, the abstract domain Σ_T is forced to contain two elements $\boldsymbol{c}_\top$ and $\boldsymbol{c}_\bot$, representing the entire state space and the empty state space, respectively.

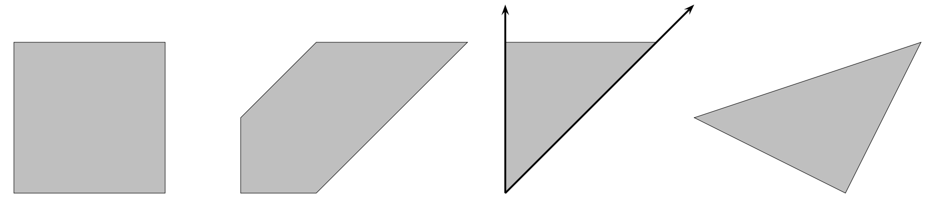
Definition 5 (Concretization function) The concretization function γ_T is defined by

$$\gamma_T(\boldsymbol{c}) \equiv \begin{cases} \mathit{false} & \mathit{if}\ \exists c_i = -\infty\ \mathit{or}\ \boldsymbol{c} = \boldsymbol{c}_\bot, \\ \mathit{true} & \mathit{if}\ \boldsymbol{c} = \boldsymbol{c}_\top, \\ \bigwedge_{i\ s.t.\ c_i \neq \infty} (T_i\boldsymbol{x} + c_i \geq 0) & \mathit{otherwise}. \end{cases}$$

The value $c_i = \infty$ drops the i^{th} constraint from the concrete assertion, and the value $c_i = -\infty$ makes the concrete assertion *false*. We assume that the standard ordering $\leq$ on the reals has been extended such that $-\infty \leq x \leq \infty$ for all $x \in \mathcal{R}$.

Example 2. Consider the template constraint matrix

$$T = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \\ -1 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{representing the template assertions} \quad \begin{bmatrix} x + c_1 \geq 0 \\ -x + c_2 \geq 0 \\ y + c_3 \geq 0 \\ -y + c_4 \geq 0 \\ -x + y + c_5 \geq 0 \\ x - y + c_6 \geq 0 \end{bmatrix}$$



$\langle 1, 1, 1, 1, \infty, \infty \rangle$ (a) $\langle 1, \infty, 1, 4, 3, 3 \rangle$ (b) $\langle 1, \infty, \infty, \infty, 3, \infty \rangle$ (c) $\langle ? \rangle$ (d)

Fig. 1. Polyhedra (a), (b) and (c) are concretizations of the elements in the abstract domain Σ_A of Example 2, whereas (d) is not.

The concretization of the abstract element $\boldsymbol{c} : \langle \infty, 2, 3, \infty, 5, 1 \rangle$ is the assertion

$$\gamma_T(\boldsymbol{c}) : \left[-x + 2 \geq 0 \ \wedge \ y + 3 \geq 0 \ \wedge \ -x + y + 5 \geq 0 \ \wedge \ x - y + 1 \geq 0 \right] \ .$$

Figure 1 shows three polyhedra that are concretizations of elements in Σ_T, and one that is not.

Definition 6 (Abstract domain pre-order) Let $\boldsymbol{a}$, $\boldsymbol{b} \in \Sigma_T$,

$$\boldsymbol{a} \preceq \boldsymbol{b} \quad \textit{iff} \quad \gamma_T(\boldsymbol{a}) \models \gamma_T(\boldsymbol{b}).$$

Also $\boldsymbol{a} \sim_{\preceq} \boldsymbol{b}$ iff $\boldsymbol{a} \preceq \boldsymbol{b}$ and $\boldsymbol{b} \preceq \boldsymbol{a}$. We set $\bot = \langle -\infty, \ldots, -\infty \rangle$ and $\top = \langle \infty, \ldots, \infty \rangle$, and for T empty, $\bot = \boldsymbol{c}_\bot$ and $\top = \boldsymbol{c}_\top$. Note that $\gamma_T(\bot) = \mathit{false}$ and $\gamma_T(\top) = \mathit{true}$.

The abstraction function α_T maps sets of states into vectors $\boldsymbol{c}$ in Σ_T. Since we restrict ourselves to linear transition systems, we may assume that sets of states can be described by linear assertions $\varphi : \ A\boldsymbol{x} + \boldsymbol{b} \geq \boldsymbol{0}$. Ideally, the value of $\alpha_T(\varphi)$ should be the vector $\boldsymbol{c}$ that represents the smallest polyhedron with shape determined by T, that subsumes φ. Thus, α_T should compute a $\preceq$-minimal $\boldsymbol{c}$ such that

$$A\boldsymbol{x} + \boldsymbol{b} \geq 0 \models T\boldsymbol{x} + \boldsymbol{c} \geq \boldsymbol{0}.$$

To begin with, if φ is unsatisfiable, we set $\boldsymbol{c} = \bot$.

If φ is satisfiable, we use linear programming to determine a suitable $\boldsymbol{c}$. Consider each half space of the form $T_i\boldsymbol{x} + c_i \geq 0$. We wish to ensure that

$$A\boldsymbol{x} + \boldsymbol{b} \geq 0 \ \models \ T_i\boldsymbol{x} + c_i \geq 0.$$

Applying Farkas Lemma, we obtain,

$$A\boldsymbol{x} + \boldsymbol{b} \geq \boldsymbol{0} \models T_i\boldsymbol{x} + c_i \geq 0 \quad \text{iff} \quad (\exists \ \boldsymbol{\lambda} \geq 0) \ A^{\mathrm{T}}\boldsymbol{\lambda} = T_i \ \wedge \ \boldsymbol{b}^{\mathrm{T}}\boldsymbol{\lambda} \leq c_i.$$

To find the smallest c_i that satisfies the requirements above, we solve the LP problem

$$\Psi : \ \boldsymbol{\lambda} \geq 0 \ \wedge \ A^{\mathrm{T}}\boldsymbol{\lambda} = T_i \ \text{ with objective function } \ \boldsymbol{b}^{\mathrm{T}}\boldsymbol{\lambda} \ .$$

If Ψ has a solution u, c_i is set to u. If Ψ has no solutions, c_i is set to ∞. The third case, where c_i is unbounded, does not occur if $A\boldsymbol{x} + \boldsymbol{b} \geq \mathbf{0}$ is satisfiable.

Claim. For satisfiable φ, then c_i cannot be unbounded in Ψ.

Proof. If c_i were unbounded, then appealing to the soundness of Farkas Lemma leads us to the conclusion that $A\boldsymbol{x} + \boldsymbol{b} \geq 0 \models T_i\boldsymbol{x} + c_i \geq 0$ for all $c_i \leq u$, for some constant u. If φ were satisfiable, then some point $\boldsymbol{x_0}$ satisfies it. Therefore, $(\boldsymbol{x} = \boldsymbol{x_0}) \models \varphi \models T_i\boldsymbol{x} + c_i \geq 0$. Setting c_i to any value strictly less than $T_i\boldsymbol{x_0}$ and u, yields $-1 \geq 0$ and hence, a contradiction.

Example 3. Consider the assertion

$$\varphi : \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 1 \\ 1 & -1 \end{pmatrix}}_{A} \begin{pmatrix} x \\ y \end{pmatrix} + \underbrace{\begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}}_{\boldsymbol{b}} \geq \mathbf{0} \ .$$

The abstraction $\alpha_T(\varphi)$ mapping φ into the abstract domain of Example 2 is computed by solving six LP problems. For example, c_2 is computed by solving

$$\underbrace{\begin{pmatrix} 1 & 0 & -1 & 1 \\ 0 & 1 & 1 & -1 \end{pmatrix}}_{A^{\mathrm{T}}} \underbrace{\begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{pmatrix}}_{\boldsymbol{\lambda}} = \underbrace{\begin{pmatrix} -1 \\ 0 \end{pmatrix}}_{T_2} \text{ with objective function } \quad \lambda_3 + \lambda_4 \ .$$

The problem has no solutions, yielding the value ∞ for c_2. The value for c_6 is computed by solving the same problem, replacing $T_2 : \ (-1\ 0)^{\mathrm{T}}$ by $T_6 : \ (1\ -1)^{\mathrm{T}}$. This problem yields an optimal solution $c_6 = 1$. Solving all the six problems produces $\alpha_T(\varphi) = \langle 0, \infty, 0, \infty, 1, 1\rangle$.

Definition 7 (Abstraction function) Let φ be the linear assertion $A\boldsymbol{x}+\boldsymbol{b} \geq \mathbf{0}$. Given a nonempty TCM T, the function α_T assigns to φ the value $\boldsymbol{c} = \langle c_1, \ldots, c_m\rangle$, such that

$$c_i = \begin{cases} -\infty & \textit{if } \varphi \textit{ is unsatisfiable}, \\ \textit{min. } \boldsymbol{b}^{\mathrm{T}}\boldsymbol{\lambda}, \ \textit{s.t.} \ \underbrace{\boldsymbol{\lambda} \geq 0 \wedge A^{\mathrm{T}}\boldsymbol{\lambda} = T_i}_{\Psi_i} & \textit{if } \Psi_i \textit{ is feasible}, \\ \infty & \textit{if } \Psi_i \textit{ is infeasible} \ . \end{cases}$$

For an empty TCM T, we set $\alpha_T(\varphi) = \boldsymbol{c}_\perp$ if φ is unsatisfiable, and $\alpha_T(\varphi) = \boldsymbol{c}_\top$ otherwise.

Lemma 1 (Abstraction Lemma). *The functions α_T and γ_T form a Galois connection, that is, (1) for all linear assertions φ and abstract elements $\boldsymbol{a} \in \Sigma_A$, $\alpha_T(\varphi) \preceq \boldsymbol{a}$ iff $\varphi \models \gamma_T(\boldsymbol{a})$. (2) Furthermore, for nonempty T, if $\varphi \models \gamma_T(\boldsymbol{a})$ then $\alpha_T(\varphi) \leq \boldsymbol{a}$. That is, $\alpha_T(\varphi)$ is minimal with respect to the standard order $\leq$ on vectors.*

Proof. For empty TCM T, both parts follow easily from Definitions 5, 6, and 7. For the remainder, assume T is nonempty.

(1) ($\Rightarrow$) Assume $\alpha_T(\varphi) \preceq \boldsymbol{a}$. If φ is unsatisfiable, then trivially, $\varphi \models \gamma_T(\boldsymbol{a})$. Otherwise, let $\alpha_T(\varphi) = \boldsymbol{c}$. By Def. 7 and the soundness of Farkas Lemma, $\varphi \models T_i\boldsymbol{x} + c_i \geq 0$, for each $c_i \neq \infty$. Therefore, $\varphi \models \gamma_T(\boldsymbol{c})$. By Def. 6 and by assuming $\boldsymbol{c} \preceq \boldsymbol{a}$, we obtain $\varphi \models \gamma_T(\boldsymbol{c}) \models \gamma_T(\boldsymbol{a})$.

($\Leftarrow$) Assume $\varphi \models \gamma_T(\boldsymbol{a})$. If φ is unsatisfiable, $\alpha_T(\varphi) = \bot$ and hence trivially $\alpha_T(\varphi) \preceq \boldsymbol{a}$. Otherwise let $\alpha_T(\varphi) = \boldsymbol{c}$. By Def 5, $\varphi \models \bigwedge_{a_i \neq \infty}(T_i\boldsymbol{x} + a_i \geq 0)$, and hence for arbitrary i such that $a_i \neq \infty$, $\varphi \models T_i\boldsymbol{x} + a_i \geq 0$. By Def 7 and the completeness of Farkas Lemma, both c_i and a_i belong to the (nonempty) feasible set of the linear program generated for the implication $\varphi \models T_i\boldsymbol{x} + c_i \geq 0$. Therefore, by optimality of c_i, $c_i \leq a_i$, and hence, $T_i\boldsymbol{x} + c_i \geq 0 \models T_i\boldsymbol{x} + a_i \geq 0$. and hence by Def 6, $\boldsymbol{c} \preceq \boldsymbol{a}$. In fact, we have also established that $\boldsymbol{c} \leq \boldsymbol{a}$.

(2) This follows directly from the second part of (1).

The abstract domain Σ_T is redundant. It contains multiple elements that map to the same concrete element. We eliminate this redundancy by choosing a *canonical element* $\boldsymbol{c}_{min}$ for each equivalence class $[\boldsymbol{c}]$ of the relation $\sim_{\preceq}$.

Example 4. Consider the abstract domain from Example 2. The elements $\boldsymbol{a_1} = \langle 1,1,1,1,2,2\rangle$, $\boldsymbol{a_2} = \langle 1,1,1,1,3,3\rangle$, and $\boldsymbol{a_3} : \langle 1,1,1,1,\infty,\infty\rangle$ all map to the rectangle described by $-1 \leq x, y \leq 1 \;\wedge\; -2 \leq x - y \leq 2$, and thus $\boldsymbol{a_1} \sim_{\preceq} \boldsymbol{a_2} \sim_{\preceq} \boldsymbol{a_3}$. The reason is that the last two constraints, on $x - y$, are redundant in all these elements. In fact any abstract element $\langle 1,1,1,1,x,y\rangle$, $x \geq 2,\ y \geq 2$ belongs to the same equivalence class.

Definition 8 (Canonical element) Let Σ_T be an abstract domain with ordering $\preceq$. Given an equivalence class $[\boldsymbol{c}]$ of $\sim_{\preceq}$, its canonical element, denoted $can(\boldsymbol{c})$ is defined as $can(\boldsymbol{c}) = \alpha_T(\gamma_T(\boldsymbol{c}))$.

We need to show that $can(\boldsymbol{c})$ belongs to the equivalence class of $\boldsymbol{c}$, and also that the canonical element is unique.

Claim. (1) $can(\boldsymbol{c}) \sim_{\preceq} \boldsymbol{c}$, (2) for any $\boldsymbol{a}$, such that $\boldsymbol{a} \sim_{\preceq} \boldsymbol{c}$, $can(\boldsymbol{c}) \leq \boldsymbol{a}$, (3) $can(\boldsymbol{c})$ is unique, i.e., if $\boldsymbol{a} \sim \boldsymbol{c}$, then $can(\boldsymbol{c}) = can(\boldsymbol{a})$.

Proof. This follows directly from Lemma 1.

Example 5. For the abstract domain of Example 2, $\langle 1,1,1,1,2,2\rangle$ is the canonical element for the equivalence class represented by

$$[\langle 1,1,1,1,2,2\rangle] = \{\langle 1,1,1,1,x,y\rangle \;\mid\; x, y \geq 2\}$$

Computation of the greatest lower bound of two canonical elements in Σ_T for nonempty T consists of taking the entrywise minimum and canonizing the result.

Definition 9 (Intersection) Let $\boldsymbol{a}$, $\boldsymbol{b}$ be two canonical elements of Σ_T. For T nonempty, $\boldsymbol{a} \sqcap \boldsymbol{b} = can(\langle \min(a_1, b_1), \ldots, \min(a_m, b_m)\rangle$, where $\min(x, y)$ is defined as the minimum under the $\leq$ relation. For T empty we define the intersection operation on the elements $\top$ and $\bot$ in the standard fashion.

The following example shows that $\langle \min(a_1, b_1), \ldots, \min(a_m, b_m)\rangle$ is not necessarily a canonical element.

Example 6 (Failure of Canonicity). Consider the abstract domain Σ_T with template constraint matrix $T = (1 \quad -1)^{\mathrm{T}}$, representing the template assertions $x + c_1 \geq 0$ and $-x + c_2 \geq 0$. The entrywise minimum of the elements $\langle 1, 2\rangle$, and $\langle 5, -2\rangle$ is $\langle 1, -2\rangle$. Verify that $\gamma_T(\langle 1, -2\rangle) = \mathit{false}$, and hence $\mathit{can}(\langle 1, -2\rangle) = \bot$.

Claim. Let $\boldsymbol{m} = \boldsymbol{a_1} \sqcap \boldsymbol{a_2}$. Then (1) $\boldsymbol{m} \preceq \boldsymbol{a_{\{1,2\}}}$ and (2) for any $\boldsymbol{b} \preceq \boldsymbol{a_{\{1,2\}}}$, it follows that $\boldsymbol{b} \preceq \boldsymbol{m}$

Proof. If T is empty, $\boldsymbol{a_1} = \bot$, or $\boldsymbol{a_2} = \bot$, both parts hold immediately.

(1) If $\boldsymbol{m} = \bot$ then the first part holds immediately. If $\boldsymbol{m} \neq \bot$, then we show that $\gamma_T(\boldsymbol{m}) \models \gamma_T(\boldsymbol{a_1})$. Since $\boldsymbol{m} = \mathit{can}(\min(\boldsymbol{a_1}, \boldsymbol{a_2}))$, for each row i, $m_i \leq \min(a_{1i}, a_{2i})$. If $a_{1i} \neq \infty$, then $m_i \neq \infty$. Therefore, $\gamma_T(\boldsymbol{m}) \models T_i\boldsymbol{x} + m_i \geq 0 \models T_i\boldsymbol{x} + a_{1i} \geq 0$. Thus, $\gamma_T(\boldsymbol{m}) \models \gamma_T(\boldsymbol{a_1})$, leading to $\boldsymbol{m} \preceq \boldsymbol{a_1}$. Similarly, $\boldsymbol{m} \preceq \boldsymbol{a_2}$.

(2) Let $\boldsymbol{b} \preceq \boldsymbol{a_{\{1,2\}}}$ and $\boldsymbol{a_{\{1,2\}}} \neq \bot$. For each i, such that $a_{1i} \neq \infty$, $\gamma_T(\boldsymbol{b}) \models T_i\boldsymbol{x} + a_{1i} \geq 0$. Similarly, if $a_{2i} \neq \infty$, then $\gamma_T(\boldsymbol{b}) \models T_i\boldsymbol{x} + a_{2i} \geq 0$. Therefore, for each $m_i \neq \infty$, there are four cases to consider depending on $a_{1i} \neq \infty$, $a_{2i} \neq \infty$. In either case, $\gamma_T(\boldsymbol{b}) \models T_i\boldsymbol{x} + \min(a_{1i}, a_{2i}) \geq 0$. Therefore $\boldsymbol{b} \preceq \min(\boldsymbol{a_1}, \boldsymbol{a_2}) \preceq \boldsymbol{m}$.

Computation of the lowest upper bound of two canonical elements consists of taking the entrywise maximum, and is guaranteed to result in a canonical element.

Definition 10 (Union) Let $\boldsymbol{a}$, $\boldsymbol{b}$ be two canonical elements of Σ_T. For T nonempty, $\boldsymbol{a} \sqcup \boldsymbol{b} = \langle \max(a_1, b_1), \ldots, \max(a_m, b_m)\rangle$ For T empty the union is the usual result for $\top$ and $\bot$.

Claim. Let $\boldsymbol{m} = \boldsymbol{a_1} \sqcup \boldsymbol{a_2}$. Then (1) $\boldsymbol{a_{\{1,2\}}} \preceq \boldsymbol{m}$; (2) if for some $\boldsymbol{b}$, $\boldsymbol{a_{\{1,2\}}} \preceq \boldsymbol{b}$, it follows that $\boldsymbol{m} \preceq \boldsymbol{b}$; and (3) $\boldsymbol{m}$ is canonical

Proof. Proofs for parts (1), (2) are similar to the proof for intersection. For part (3), assume otherwise. Then there exists a vector $\boldsymbol{b} \sim \boldsymbol{m}$, and some position j such that $b_j < m_j$. Assume w.l.o.g., that $a_{1j} \leq a_{2j} = m_j$. Let $\boldsymbol{a_2}'$ be the vector $\boldsymbol{a_2}$ with a_{2j} replaced by b_j. It follows immediately, that $\boldsymbol{a_2}' \preceq \boldsymbol{a_2}$. $\gamma_T(\boldsymbol{a_2}) \models \gamma_T(\boldsymbol{m}) \models T_j\boldsymbol{x} + b_j \geq 0$, therefore $\boldsymbol{a_2} \preceq \boldsymbol{a_2}'$, and consequently, $\boldsymbol{a_2} \sim \boldsymbol{a_2}'$. Thus $\boldsymbol{a_2}$ fails to be canonical in this case, contradicting our assumptions.

Claim. Let $\boldsymbol{a}$, $\boldsymbol{b}$ be two canonical elements. Then $\boldsymbol{a} \preceq \boldsymbol{b}$ iff for each i, $a_i \leq b_i$.

Proof. This follows directly from the two claims above, using the fact that $\boldsymbol{a} \preceq \boldsymbol{b}$ iff $\boldsymbol{a} \sqcup \boldsymbol{b} \sim_{\preceq} \boldsymbol{b}$, along with the property that if two canonical forms are equivalent then they are identical.

3.2 Analysis Algorithm

Traditionally forward propagation is performed entirely in the abstract domain until convergence, and the resulting fixed point is concretized. Our analysis algorithm performs the analysis in *multiple abstract domains*: one domain per program location. This allows for tailoring the template constraint matrix to the assertions that are likely to hold at that location. It also complicates the presentation of the *post* operation.

Let $\Psi : \langle L, \mathcal{T}, \ell_0, \Theta \rangle$ be an LTS over a set of variables V. Let each location $\ell \in L$ be associated with an abstract domain Σ_ℓ parameterized by the template constraint matrix T_ℓ, with k_ℓ template rows. The objective is to construct an abstract invariant map η that maps each location $\ell \in L$ to a (canonical) element in the abstract domain Σ_ℓ.

We construct this invariant map by forward propagation as follows. The starting point is the map η^0 that assigns to the initial location, ℓ_0, the abstract value of the initial condition, that is $\eta^0(\ell_0) = \alpha_{\ell_0}(\Theta)$, and the element $\bot$ to all other locations.

Example 7. Consider the LTS from Example 1. The associated domain of the single location ℓ_0 has template constraint matrix

$$\begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \begin{array}{c}\text{representing the} \\ \text{template assertions}\end{array} \quad \begin{bmatrix} x & & + c_1 \geq 0 \\ -x & & + c_2 \geq 0 \\ x & + y & + c_3 \geq 0 \\ x & - y & + c_4 \geq 0 \end{bmatrix}$$

Using this template, the initial condition, $\Theta :\ x = 0 \ \wedge\ y = 0$ is abstracted to $\eta^0(\ell_0) \ = \ \langle 0, 0, 0, 0 \rangle$.

The postcondition operator for a transition leading from location ℓ_i to location ℓ_j computes the element $\boldsymbol{c}$ in Σ_{ℓ_j} that represents the states that can be reached from states represented by the current value of η at ℓ_i. More formally,

Definition 11 (Postcondition operator) Given $\tau :\ \langle \ell_i, \ell_j, \rho_\tau \rangle$, then

$$post(\eta(\ell_i), \tau) = \begin{cases} \bot & \eta(\ell_i) = \bot \\ \alpha_j(\gamma_i(\eta(\ell_i) \ \wedge\ \rho_\tau)) & \textit{otherwise} \end{cases}$$

where α_j is the abstraction function for Σ_{ℓ_j} and γ_i is the concretization function of Σ_{ℓ_i}.

Let T_{ℓ_i} and T_{ℓ_j} be the template constraint matrices for locations ℓ_i and ℓ_j, respectively. Let ρ_τ be $\xi \wedge \boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{b}$. If $post(\eta(\ell_i), \tau) = \boldsymbol{c}$, we require that

$$\begin{array}{l}(T_{\ell_i}\boldsymbol{x} + \eta(\ell_i) \geq 0) \ \wedge\ \xi \wedge \boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{b} \ \models\ T_{\ell_j}\boldsymbol{x}' + \boldsymbol{c} \geq 0, \ \textit{equivalently}, \\ (T_{\ell_i}\boldsymbol{x} + \eta(\ell_i) \geq 0) \ \wedge\ \xi \models (T_{\ell_j}A)\boldsymbol{x} + (T_{\ell_j}\boldsymbol{b} + \boldsymbol{c})\end{array}$$

In practice, we precompute the TCM $T' = T_{\ell_j}A$ for each transition. We then abstract the assertion $\gamma_j(\eta(\ell_i)) \ \wedge\ \xi$ using this TCM. Care should be taken to subtract $T_{\ell_j}\boldsymbol{b}$ from the result of the abstraction. This yields the post-condition at location ℓ_j w.r.t. transition τ. Note that this technique is also applied to self-looping transitions. Therefore, labeling each location with a different template complicates our presentation but not the complexity of the procedure.

Example 8. Consider the map $\eta^0(\ell_0) = \langle 0, 0, 0, 0\rangle$ from Example 7 and the transition $\tau_1 = \big\langle \ell_0, \ell_0, \big[x' = x + 2y \ \wedge \ y' = 1 - y\big]\big\rangle$. For this transition, $\xi = \mathit{true}$,

$$A = \begin{pmatrix} 1 & 2 \\ 0 & -1 \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{b} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

We compute $T' = TA$ and $\boldsymbol{a} = T\boldsymbol{b}$ for performing the abstraction. Abstracting $\gamma_T(\eta(\ell_0)) \equiv (x = y = 0)$, w.r.t T' yields the result $\langle 0, 0, 0, 0\rangle$. Subtracting $\boldsymbol{a}$ from this yields, $\langle 0, 0, -1, 1\rangle$, which is the required post-condition.

Using the postcondition the map at step $i > 0$ is updated in the standard fashion, as follows:

$$\eta^{i+1}(\ell_n) = \eta^i(\ell_n) \sqcup \left(\bigsqcup_{\tau:\langle \ell_m, \ell_n, \rho\rangle} \mathit{post}(\eta^i(\ell_m), \tau) \right)$$

This process does not necessarily terminate. Termination can be ensured by a form of widening that is a natural generalization of widening in the interval and octagon domain. At each location we limit the number of updates to each parameter to a fixed number. If that number is exceeded for a particular parameter, we impoverish the abstract domain at that location by removing the corresponding constraint from the TCM. Clearly this guarantees termination, since for each TCM the number of constraints is finite.

Remark The reason that we remove the constraint from the TCM, rather than set the parameter to ∞, is that the latter may lead to infinite ascending chains. The problem, as pointed out by Miné [13, 12], is that when a parameter c_i is set to ∞, subsequent canonization may set c_i back to a finite value, effectively bringing back the corresponding constraint.

Example 9. Figure 2 shows the results of applying the algorithm to the LTS in Example 1. The maximum number of updates to any template constraint expression is set to 3. The Figure shows that only three constraints survive, corresponding to the invariants $x \geq 0 \ \wedge \ x + 2y \geq 0 \ \wedge x + y \geq 0$.

Instead of directly removing a constraint from the TCM if the number of updates has exceeded the threshold, a more elegant solution is to use a *Local Widening* operator, similar to the widening-upto operator introduced in [8]. Let $T_{\ell,i}\boldsymbol{x} + c_i \geq 0$ be one of the template constraints at location ℓ, such that the number of updates to c_i has exceeded the set threshold. Let τ be an incoming transition at location ℓ. The *local value* of c_i w.r.t τ is obtained by computing the minimum c_i for which $\rho_\tau \models T_{\ell,i}\boldsymbol{x'} + c_i \geq 0$ holds. Assuming that τ can

Template	*Iteration num.*						
	1	2	3	4	5	6	7
$y + c_1$	0	0	2	4	6	x	x
$-y + c_2$	0	3	5	7	x	x	x
$x + c_3$	0	0	0	0	0	0	0
$-x + c_4$	0	1	8	x	x	x	x
$x - y + c_5$	0	2	3	4	x	x	x
$x + y + c_6$	0	0	0	0	0	0	0
$-x - y + c_7$	0	4	8	x	x	x	x
$y - x + c_8$	0	0	9	16	x	x	x
$x + 2y + c_9$	0	0	0	0	0	0	0
$-x - 2y + c_{10}$	0	7	12	17	x	x	x
$x + 3y + c_{11}$	0	0	0	0	1	∞	∞
$-x - 3y + c_{12}$	0	10	17	24	x	x	x

Fig. 2. A run of the invariant generation algorithm

be executed for some state, the corresponding LP problem either shows optimal solution $b_{i,\tau}$, or is infeasible, in which case the local value is set to ∞. As a result, if b_i is the maximum among all the local values of all the transitions τ with target location ℓ, the assertion $T_{\ell,i}\boldsymbol{x} + b_i \geq 0$ is a *local invariant* at ℓ, and c_i can be set to b_i instead of ∞. Thus, the local widening operator computes the local value of an expression instead of dropping the expression. In this case the computed value is *frozen*, and further updates to it are disallowed.

Example 10. Consider the template expression $-x + c \geq 0$ and the transition $\tau : \; x \leq 3 \;\wedge\; x' = x + 2$. The local value of c w.r.t. τ is 5, since $\tau \models -x' + 5 \geq 0$.

3.3 Template Formation

The algorithm presented so far has assumed the presence of a template constraint matrix for every location. In this section we propose some strategies for constructing these templates.

A first source of expressions is the description of the transition system itself: expressions in the initial condition and transition guards are likely to be invariants for some constant values. A second source are expressions present in target properties. A third source are expressions of a certain form such as intervals, $x_i \leq c$, $x_i \geq c$, which are often useful in applications involving array bounds and pointer safety, or octagons, $\pm x_i \pm x_j + c \geq 0$ for each pair of variables [13]. However, these expressions, albeit good candidates, cannot be the only expressions. The reason is that they seldom are inductive by themselves: they need support.

Example 11. Consider the LTS from Example 1 and the assertion $x \geq 0$. Although Example 9 showed that $x \geq 0$ is an invariant, it is not preserved by the transition $\tau_1 : \; x' = x + 2y, y' = 1 - y$. However, $x + 2y \geq 0 \wedge x \geq 0$ is inductive. We call $x + 2y \geq 0$ the *support expression* for $x \geq 0$.

Type	Original		Support	
	TCM	Template expression	TCM	Template expression
bound	$1 \quad 0$	$x + c_1 \geq 0$	$1 \quad 2$	$x + 2y + c_2 \geq 0$
bound	$-1 \quad 0$	$-x + c_3 \geq 0$	$-1 \quad -2$	$-x - 2y + c_4 \geq 0$
bound	$0 \quad 1$	$y + c_5 \geq 0$	$0 \quad -1$	$- y + c_6 \geq 0$
octagon	$1 \quad 1$	$x + y + c_7 \geq 0$	$1 \quad 1$	$x + y + c_7 \geq 0$
octagon	$1 \quad -1$	$x - y + c_8 \geq 0$	$1 \quad 3$	$x + 3y + c_9 \geq 0$
octagon	$-1 \quad 1$	$-x + y + c_{10} \geq 0$	$-1 \quad -3$	$-x - 3y + c_{11} \geq 0$
octagon	$-1 \quad -1$	$-x - y + c_{12} \geq 0$	$-1 \quad -1$	$-x - y + c_{12} \geq 0$

Fig. 3. Support vectors for bound and octagon expressions

Definition 12 (Support vector) Given a coefficient vector $\boldsymbol{a}$ and a transition $\tau : \langle \ell_i, \ell_j, \xi \wedge \boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{b} \rangle$, the coefficient vector $(A^{\mathrm{T}}\boldsymbol{a})^{\mathrm{T}}$ is called the support vector for $\boldsymbol{a}$ with respect to τ.

Example 12. The support vector for the vector $\langle 1, -1 \rangle$ corresponding with $x - y$ from Example 7, under the update $x' = x + 2y,\ y' = 1 - y$ is $\langle 1, 3 \rangle$ corresponding with $x + 3y$. The table in Figure 3 shows the support vectors and their corresponding template expressions for the interval and octagon expressions in Example 9 with respect to τ_1. Every vector is its own support with respect to transition τ_2. In this case, the table is closed under computing support vectors.

Support vectors are computed for each location. Given template constraint matrices T_{ℓ_i} and T_{ℓ_j} and a transition $\tau : \langle \ell_i, \ell_j, \rho_\tau \rangle$, then a support vector for location ℓ_j with respect to τ is computed from a row of T_{ℓ_j} and ρ_τ and added as a row to T_{ℓ_i}. Note the similarity to computing weakest preconditions.

4 Performance

Complexity The complexity of our algorithm is polynomial in the size of the program and the size of the template constraint matrix. Consider a transition system with n variables, $|L|$ locations and $|T|$ transitions, and assume that each location is labeled with an $m \times n$ template constraint matrix. Let k be the maximum number of updates allowed to the abstract invariant map. Then the number of post-condition computations is bounded by $(k+1)m|L||T|$. Each post-condition computation requires m LP queries, one for each row in the template constraint matrix, and thus the total number of LP queries is $O(km^2|L||T|)$.

In practice, the number of updates to reach convergence is much less than $(k + 1)m|L||T|$. In addition, the number of LP queries can be reduced further by skipping post-condition computations for transitions whose prelocation assertions did not change.

Practial Performance We have implemented our algorithm and applied it to several benchmark programs. The results are shown in Figure 4. Our implementation is based on the GNU Linear programming kit, which uses the SIMPLEX

Program			Template		Statistics				
name	$\lvert L \rvert$	$\lvert T \rvert$	#t	#s	t(sec)	t_{lp}(sec)	# LPS	#avg.	#dim.
MCC91 (3)	1	2	11	0	0.05	0.01	227	1.5	15 (20)
TRAINHPR97(3)	4	12	58	3	0.1	0.02	673	0.9	18(25)
BERKELEY(4)	1	3	63	16	0.23	0.11	1,632	1.36	64(96)
DRAGON(5)	1	12	129	157	3.94	2.38	11,426	3.23	202 (298)
HEAPSORT(5)	1	4	33	24	0.34	0.13	1,751	2.45	75(90)
EFM(6)	1	5	506	461	7.65	2.36	10,872	0.69	359(981)
LIFO(7)	1	10	85	79	1.87	0.91	5,401	3.37	141 (174)
CARS-MIDPT(7)	1	2	101	324	3.72	2.21	4,641	6.23	154(329)
BARBER(8)	1	12	128	0	1.97	0.83	9,210	1.96	124(141)
SWIM-POOL(9)	1	6	104	0	0.56	0.27	2,710	2.11	97(118)
TTP(9)	4	20	3,555	127	62.8	40.9	61,263	4.41	574(1032)
REQ-GRANT(11)	1	8	221	18	2.96	1.41	8,635	2.10	241(255)
CONSPROT(12)	2	14	533	40	4.88	2.00	12,487	1.83	266(286)
CSM(13)	1	8	313	73	9.65	5.21	14,890	3.69	380(414)
C-PJAVA(16)	1	14	453	93	35.16	15.19	33,288	5.00	433(567)
CONSPROD(18)	1	14	529	96	38.72	19.43	35,797	5.17	468(663)
INCDEC(32)	1	28	961	267	287.54	110.27	103,841	6.57	877(1294)
MESH2X2(32)	1	32	438	0	43.9	17.5	52,622	4.53	390(506)
BIGJAVA(44)	1	37	864	376	331.98	117.68	122,643	5.25	1018 (1280)
MESH3X2(52)	1	54	1133	0	432.85	192.15	216,600	6.70	930(1241)

Fig. 4. Experimental results for benchmark examples. All timings were measured on an Intel Xeon processor running linux 2.4, with 2GB RAM.

algorithm for linear programming [10]. The library uses floating point arithmetic. Soundness is maintained by careful rounding, and checking the obtained invariants using the exact arithmetic implemented in the polyhedral library PPL [1].

The benchmark programs were taken from the related work, mostly from the FAST project [2]. Many of these programs (eg., BERKELEY, DRAGON, TTP and CSM) are models of bus and network protocols. Other programs, including BIGJAVA, C-PJAVA and HEAPSORT, were obtained by abstracting java programs. Some programs are academic examples from the Petri net literature (eg., SWIM-POOL, EFM, MESHIXJ). These programs, ranging in size from 4 to 52 variables, exhibit complex behaviours and require non-trivial invariants for their correctness proofs. Figure 4 shows for each program the number of variables (next to the name in parentheses), the number of locations ($|L|$) and transitions ($|T|$).

The templates for these programs were obtained in two ways: they were generated from user-defined *patterns* or automatically derived from the initial condition and the transition guards. An example of a user-defined pattern is: "`%i + 2 * %j + 3 * %k` ". It generates all constraints of the form $x_i + 2x_j + 3x_k +$

$b_{ijk} \geq 0$, for all combinations (x_i, x_j, x_k) of system variables. In many cases the patterns were suggested by the target property. For instance the target property $x \leq K$ for some variable x and constant K, suggests the patterns `-%i`, `-%i -%j` and so on. The columns "#t" and "#s" in Figure 4 show the number of template constraints considered initially, and the number of support constraints generated for these initial constraints, respectively. Thus the total number of constraints is the sum of these two values. For each program, the maximum number of updates to each constraint was set to 3.

The statistics part in Figure 4 shows the performance of our algorithm in terms of computation time and number of LP queries solved. The first two columns (t(sec) and t_{lp}) show the total time needed to reach convergence and the time spent by the LP solver, respectively. The last three columns show the number of LP instances solved, the average number of SIMPLEX iterations for each LP call, and the the maximum (and average between parentheses) dimensionality of each LP problem. The memory used ranged from KBs for the smaller examples to 50 MB for BIGJAVA and 67MB for MESH3X2.

Invariants. The invariants obtained for the benchmark programs were of mixed quality. On one hand, the pattern-generated constraints produced invariants that were able to prove the target properties for most examples including the CSM and BIGJAVA. On the other hand, we were unable to prove the desired properties for examples like INCDEC and CONSPROD. In general, like with polyhedra, our technique fails in cases where non-convex and non-linear invariants are required. For all programs the propagation converged within 10 iterations, which is much faster than the theoretical maximum.

5 Conclusions

In this paper, we have demonstrated an efficient algorithm for computing invariants by applying abstract interpretation on a domain that is less powerful than that of polyhedra but more general than related domains like intervals, octagons and the very recent octahedra. In theory, we have thus generalized the previous results and appealed to the complexity of linear programming to show that all of these analyses can be performed in polynomial time. In practice, we have exploited the power of LP solvers to provide time and space-efficient alternatives to polyhedra. We have shown through our benchmark examples that our method is scalable to large examples and has the potential of scaling to even larger examples through a wiser choice of templates. Our support assertion generation greatly improves the ability of our algorithm to infer non-trivial invariants, and exploits the fact that we can support arbitrary coefficients in our assertions.

Future extensions to this work need to consider many issues both theoretical and practical. The analysis can be performed on non-canonical elements. This can greatly simplify the post-condition computation but complicate inclusion checks. Preprocessing LP calls using an equality simplifier could reduce the dimensionality of each call. Possible extensions include the use of semi-definite programming to extend the method to non-linear systems and non-linear templates. The work of Parillo et al. gives us a direct extension of Farkas Lemma for the non-linear case [14].

Acknowledgements We would like to thank the anonymous referees for their comments and suggestions. We are also grateful to the developers of GLPK [10] and PPL [1] for making their tools public, and hence making this study possible. Many thanks to Aaron Bradley, Michael Colón, César Sánchez and Matteo Slanina for their comments and suggestions.

References

1. Bagnara, R., Ricci, E., Zaffanella, E., and Hill, P. M. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In *Static Analysis Symposium* (2002), vol. 2477 of *LNCS*, Springer-Verlag, pp. 213–229.
2. Bardin, S., Finkel, A., Leroux, J., and Petrucci, L. FAST: Fast accelereation of symbolic transition systems. In *Computer-aided Verification* (July 2003), vol. 2725 of *LNCS*, Springer-Verlag.
3. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., and Rival, X. A static analyzer for large safety-critical software. In *ACM SIGPLAN PLDI'03* (June 2003), vol. 548030, ACM Press, pp. 196–207.
4. Clarisó, R., and Cortadella, J. The octahedron abstract domain. In *Static Analysis Symposium* (2004), vol. 3148 of *LNCS*, Springer-Verlag, pp. 312–327.
5. Cousot, P., and Cousot, R. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming* (1976), Dunod, Paris, France, pp. 106–130.
6. Cousot, P., and Cousot, R. Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Principles of Programming Languages* (1977), pp. 238–252.
7. Cousot, P., and Halbwachs, N. Automatic discovery of linear restraints among the variables of a program. In *ACM Principles of Programming Languages* (Jan. 1978), pp. 84–97.
8. Halbwachs, N., Proy, Y., and Roumanoff, P. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design 11*, 2 (1997), 157–185.
9. Karr, M. Affine relationships among variables of a program. *Acta Inf. 6* (1976), 133–151.
10. Makhorin, A. The GNU Linear Programming Kit, 2000. `http://www.gnu.org/software/glpk/glpk.html`.
11. Manna, Z., and Pnueli, A. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York, 1995.
12. Miné, A. A new numerical abstract domain based on difference-bound matrices. In *PADO II* (May 2001), vol. 2053 of *LNCS*, Springer-Verlag, pp. 155–172.
13. Miné, A. The octagon abstract domain. In *AST 2001 in WCRE 2001* (October 2001), IEEE, IEEE CS Press, pp. 310–319.
14. Parrilo, P. A. Semidefinite programming relaxation for semialgebraic problems. *Mathematical Programming Ser. B 96*, 2 (2003), 293–320.
15. Schrijver, A. *Theory of Linear and Integer Programming.* Wiley, 1986.