Generation of Basic Semi-algebraic Invariants Using Convex Polyhedra*

Roberto Bagnara[1], Enric Rodríguez-Carbonell[2], and Enea Zaffanella[1]

[1] Department of Mathematics, University of Parma, Italy
{bagnara,zaffanella}@cs.unipr.it

[2] Software Department, Technical University of Catalonia, Spain
erodri@lsi.upc.edu

Abstract. A technique for generating invariant polynomial *inequalities* of bounded degree is presented using the abstract interpretation framework. It is based on overapproximating basic semi-algebraic sets, i.e., sets defined by conjunctions of polynomial inequalities, by means of convex polyhedra. While improving on the existing methods for generating invariant polynomial *equalities*, since polynomial inequalities are allowed in the guards of the transition system, the approach does not suffer from the prohibitive complexity of the methods based on quantifier-elimination. The application of our implementation to benchmark programs shows that the method produces non-trivial invariants in reasonable time. In some cases the generated invariants are essential to verify safety properties that cannot be proved with classical linear invariants.

1 Introduction

The discovery of invariant properties is at the core of the analysis and verification of infinite state systems such as sequential programs and reactive systems. For this reason, invariant generation has been a major research problem since the seventies. Abstract interpretation [11] provides a solid foundation for the development of techniques automatizing the synthesis of invariants of several classes, most significantly intervals [10], linear equalities [20] and linear inequalities [14].

For some applications, linear invariants are not enough to get a precise analysis of numerical programs and nonlinear invariants may be needed as well. For example, the ASTRÉE static analyzer, which has been successfully employed to verify the absence of run-time errors in flight control software [13], implements the ellipsoid abstract domain [7], which represents a certain class of quadratic inequality invariants. Moreover, it has been acknowledged elsewhere [28, 30] that nonlinear invariants are sometimes required to prove program properties.

As a consequence, a remarkable amount of work has been recently directed to the generation of invariant *polynomial equalities*. Some of the methods plainly

* This work has been partially supported by PRIN project "AIDA — Abstract Interpretation: Design and Applications," by the "LogicTools" project (CICYT TIN 2004-03382), and the FPU grant AP2002-3693 from the Spanish MEC.

ignore all the conditional guards [25, 27]; other methods can only consider the polynomial equalities in the guards [8, 31], whereas some other proposals [23, 26] can handle *polynomial disequalities* in guards (i.e., guards of the form $p \neq 0$ where p is a polynomial). None of the techniques previously mentioned can handle the case of *polynomial inequalities* in the guards: these are ignored to the expense of precision.

In this paper we present a method for generating conjunctions of polynomial inequalities as invariants of transition systems, which we have chosen as our programming model. The transition systems that the approach can handle admit finite conjunctions of polynomial inequalities as guards and initial conditions, as well as polynomial assignments and nondeterministic assignments where the *rvalue* is unknown (these may correspond, for instance, to the assignment of expressions that cannot be modeled by means of polynomials).

Formally, our technique is an abstract interpretation in the lattice of *polynomial cones* of bounded degree, which are the algebraic structures analogous to vector spaces in the context of polynomial equality invariants [8]. Intuitively, the approach is based on considering nonlinear terms as additional independent variables and using convex polyhedra to represent polynomial cones in this extended set of variables. In order to reduce the loss of precision induced by this overapproximation, additional linear constraints are added conservatively to the polyhedra, so as to enforce some (semantically redundant) nonlinear constraints that would be lost in the translation. The strength of the approach is that, while allowing for a much broader class of programs than linear analysis, it uses the very same underlying machinery: this permits the adoption of already existing implementations of convex polyhedra like [4], as well as the possibility of resorting to further approximations, such as *bounded differences* [1] or *octagons* [22], when facing serious scalability problems.

The rest of the paper is organized as follows. In the next subsection, related work is briefly reviewed. Section 2 gives background information on algebraic geometry, transition systems and abstract interpretation. Section 3 presents the main contribution of the paper, where it is shown how polynomial inequalities can be discovered as invariants by means of polynomial cones, represented as convex polyhedra. The experimental evaluation of our implementation of these ideas is described in Section 4. Finally in Section 5 we summarize the contributions of the paper and sketch some ideas for future work.

1.1 Related Work

To the best of our knowledge, the first contribution towards the generation of invariant polynomial inequalities is [6]. The authors consider a simple class of transition systems, where assignments are of the form $x := x + k$ or $x := k$ with $k \in \mathbb{Z}$. Such a transition system is soundly abstracted into a new one whose exact reachability set is computable and overapproximates the reachability set of the original system. Besides the fact that the programming model is more restrictive than the one used in this paper, these ideas do not seem to have undergone experimental evaluation so that, as far as we can tell, their practical value remains to be assessed.

In [19], Kapur proposes a method based on imposing that a template polynomial inequality with undetermined coefficients is invariant and solving the resulting constraints over the coefficients by real quantifier elimination. Unfortunately, the great computational complexity of quantifier elimination appears to make the method impractical: as the author reports, an experimental implementation performed poorly or did not return any answer for all the analyzed programs [D. Kapur, personal communication, 2005].

A similar idea is at the core of [9, 28], where, instead of real quantifier elimination, semidefinite programming is employed. The method, which is reported to perform rather efficiently for several interesting cases, automatically determines *one* solution to the constraint system on the template parameters. This is particularly appropriate for proving program termination because, once a class of candidate ranking functions has been chosen, any solution belonging to this class is good enough. The same approach has also been applied to the computation of invariant properties. In this case, according to [9], the one above becomes the main limitation of the method: any invariant property, even a weak one, may be obtained and it is unclear whether it is possible to drive the solver so as to produce a more precise invariant in the same class.

In [30], Sankaranarayanan et al. propose a technique for generating linear invariants by linear programming. It is based on imposing, as invariants, constraints where the coefficients of the variables are fixed *a priori*; the analysis then returns, for each such constraint, an independent term for which the constraint is indeed an invariant of the system (in the case where this is not possible, the analysis returns $\pm\infty$). A generalization of this approach for the discovery of invariant polynomial inequalities by means of semidefinite programming is sketched. Similarly, the ellipsoid abstract domain [7] allows to generate invariant quadratic inequalities with two variables by also fixing the coefficients of terms and leaving the independent term to be determined by the analysis. The approach proposed in this paper differs in that we do not need to fix any of these coefficients in advance, but rather it is the analysis itself that determines all coefficients.

2 Preliminaries

2.1 Algebraic Geometry

We denote the real numbers by $\mathbb{R}$, and the nonnegative real numbers by $\mathbb{R}_+$. A *term* in the tuple of variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ is any expression of the form $\boldsymbol{x}^{\boldsymbol{\alpha}} := x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$, where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n$. A *monomial* is an expression of the form $c \cdot \boldsymbol{x}^{\boldsymbol{\alpha}}$, simply written as $c\boldsymbol{x}^{\boldsymbol{\alpha}}$, where $c \in \mathbb{R}$ and $\boldsymbol{x}^{\boldsymbol{\alpha}}$ is a term. The *degree* of a monomial $c\boldsymbol{x}^{\boldsymbol{\alpha}}$ with $c \neq 0$ is $\deg(c\boldsymbol{x}^{\boldsymbol{\alpha}}) := \alpha_1 + \cdots + \alpha_n$; the degree of 0 is $\deg(0) := -\infty$. A *polynomial* is a finite sum of monomials. The set of all polynomials in $\boldsymbol{x}$ with coefficients in $\mathbb{R}$ is denoted by $\mathbb{R}[\boldsymbol{x}]$. In the following we will only consider polynomials in *canonical form*, meaning that all the monomials occurring in them have non-null coefficients and distinct terms. The degree of a non-null polynomial is the maximum of the degrees of its monomials. We denote by $\mathbb{R}_d[\boldsymbol{x}]$ the set of all polynomials in $\mathbb{R}[\boldsymbol{x}]$ having degree at most d. In particular, the polynomials in $\mathbb{R}_1[\boldsymbol{x}]$, i.e., having degree at most 1, are called *linear*; similarly, the polynomials in $\mathbb{R}_2[\boldsymbol{x}]$ are called *quadratic*.

A *polynomial equality* (resp., *polynomial inequality*) is a formula of the form $p = 0$ (resp., $p \geq 0$), where $p \in \mathbb{R}[\boldsymbol{x}]$. Both will be referred to as *polynomial constraints* or simply *constraints*. Given a constraint system ψ, i.e., a finite set of polynomial constraints, we define

$$\operatorname{poly}(\psi) := \bigl\{\, p \in \mathbb{R}[\boldsymbol{x}] \bigm| (p = 0) \in \psi \text{ or } (-p = 0) \in \psi \text{ or } (p \geq 0) \in \psi \,\bigr\}.$$

We will sometimes abuse notation by writing the set ψ to denote the finite conjunction of the constraints occurring in it.

The *algebraic set* defined by a finite set of polynomials $\{p_1, \ldots, p_k\} \subseteq \mathbb{R}[\boldsymbol{x}]$ is the set of points that satisfy the corresponding polynomial equalities, i.e., $\bigl\{\, \boldsymbol{v} \in \mathbb{R}^n \bigm| p_1(\boldsymbol{v}) = 0, \ldots, p_k(\boldsymbol{v}) = 0 \,\bigr\}$. Similarly, the *basic semi-algebraic set* defined by the same set of polynomials is the set of points that satisfy all the corresponding polynomial inequalities: $\bigl\{\, \boldsymbol{v} \in \mathbb{R}^n \bigm| p_1(\boldsymbol{v}) \geq 0, \ldots, p_k(\boldsymbol{v}) \geq 0 \,\bigr\}$.

2.2 Transition Systems

In this section we define our programming model: *transition systems.*

Definition 1. (Transition system.) *A* transition system $(\boldsymbol{x}, \mathcal{L}, \mathcal{T}, \mathcal{I})$ *is a tuple that consists of the following components:*

- *An n-tuple of real-valued* variables $\boldsymbol{x} = (x_1, \ldots, x_n)$.
- *A finite set $\mathcal{L}$ of* locations.
- *A finite set $\mathcal{T} \subset \mathcal{L} \times \mathcal{L} \times \wp(\mathbb{R}^n) \times \bigl(\mathbb{R}^n \to \wp(\mathbb{R}^n)\bigr)$ of* transitions. *A transition $(\ell, \ell', \gamma, \rho) \in \mathcal{T}$ consists of a* source location *$\ell \in \mathcal{L}$, a* target location *$\ell' \in \mathcal{L}$, a* guard *$\gamma \in \wp(\mathbb{R}^n)$ that enables the transition, and, finally, an* update map *$\rho \colon \mathbb{R}^n \to \wp(\mathbb{R}^n)$ that relates the values of the variables before and after the firing of the transition.*
- *A map $\mathcal{I} \colon \mathcal{L} \to \wp(\mathbb{R}^n)$ from locations to* initial conditions.

The guards, the update maps and the initial conditions are all assumed to be finitely computable.

The state of a transition system is completely characterized by the location at which control resides and by a valuation for the variables.

Definition 2. (Local and global state.) *A* local state *(at some unspecified location) is any real vector $\boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n$, interpreted as a valuation for the variables $\boldsymbol{x} = (x_1, \ldots, x_n)$: in local state $\boldsymbol{v}$, we have $x_i = v_i$ for each $i = 1$, $\ldots$, n. A* global state *is a pair $(\ell, \boldsymbol{v})$, where $\ell \in \mathcal{L}$ and $\boldsymbol{v}$ is the local state at ℓ.*

Definition 3. (Run, initial state.) *A* run *of the transition system* $(\boldsymbol{x}, \mathcal{L}, \mathcal{T}, \mathcal{I})$ *is a sequence of global states* $(\ell_0, \boldsymbol{v}_0)$, $(\ell_1, \boldsymbol{v}_1)$, $(\ell_2, \boldsymbol{v}_2)$, $\ldots$ *such that* (1) $(\ell_0, \boldsymbol{v}_0)$ *is an* initial state*, that is* $\boldsymbol{v}_0 \in \mathcal{I}(\ell_0)$*, and* (2) *for each pair of consecutive states,* $(\ell_i, \boldsymbol{v}_i)$ *and* $(\ell_{i+1}, \boldsymbol{v}_{i+1})$*, there exists a transition* $(\ell_i, \ell_{i+1}, \gamma, \rho) \in \mathcal{T}$ *that is enabled, i.e.,* $\boldsymbol{v}_i \in \gamma$*, and such that* $\boldsymbol{v}_{i+1} \in \rho(\boldsymbol{v}_i)$*.*

The fundamental notion is that of an *invariant* of a transition system:

Definition 4. (Reachable state, invariant property and map.) *A global state* $(\ell, \boldsymbol{v})$ *is called* reachable *in the transition system* $S = (\boldsymbol{x}, \mathcal{L}, \mathcal{T}, \mathcal{I})$*, if there exists a run* $(\ell_0, \boldsymbol{v}_0)$, $(\ell_1, \boldsymbol{v}_1)$, $\ldots$, $(\ell_m, \boldsymbol{v}_m)$ *of* S *such that* $(\ell, \boldsymbol{v}) = (\ell_m, \boldsymbol{v}_m)$*. We denote the set of reachable states of* S *by* $\mathrm{reach}(S)$*, and the set of (local) reachable states at location* ℓ*, i.e., those* $\boldsymbol{v}$ *such that* $(\ell, \boldsymbol{v}) \in \mathrm{reach}(S)$*, by* $\mathrm{reach}_\ell(S)$*.*

If $\boldsymbol{x} = (x_1, \ldots, x_n)$*, an* invariant property *of* S *at location* $\ell \in \mathcal{L}$ *(also called an* invariant*) is any set* $I \in \wp(\mathbb{R}^n)$ *such that* $\mathrm{reach}_\ell(S) \subseteq I$*. Finally, an* invariant map *is a map* $\mathrm{inv}\colon \mathcal{L} \to \wp(\mathbb{R}^n)$ *such that for any* $\ell \in \mathcal{L}$*,* $\mathrm{inv}(\ell)$ *is an invariant of* S *at location* ℓ*.*

In this paper we focus on a particular class of transition systems, *basic semi-algebraic transition systems*:

Definition 5. (Basic semi-algebraic transition system.) *A transition system* $(\boldsymbol{x}, \mathcal{L}, \mathcal{T}, \mathcal{I})$*, where* $\boldsymbol{x} = (x_1, \ldots, x_n)$*, is called* basic semi-algebraic *if:*

1. *for all* $(\ell, \ell', \gamma, \rho) \in \mathcal{T}$*,* γ *is a basic semi-algebraic set and there exist* $k \leq n$ *polynomials* $p_1, \ldots, p_k \in \mathbb{R}[\boldsymbol{x}]$ *and distinct indices* $i_1, \ldots, i_k \in \{1, \ldots, n\}$ *such that* $\rho(\boldsymbol{v}) = \bigl\{\, (v'_1, \ldots, v'_n) \in \mathbb{R}^n \bigm| v'_{i_1} = p_1(\boldsymbol{v}), \ldots, v'_{i_k} = p_k(\boldsymbol{v}) \,\bigr\}$ *for each* $\boldsymbol{v} \in \mathbb{R}^n$*;*
2. $\mathcal{I}(\ell)$ *is a basic semi-algebraic set, for each* $\ell \in \mathcal{L}$*.*

Notice that a basic semi-algebraic transition system can also model *nondeterministic assignments*, that is, assignments whose *rvalue* is unknown.

Example 1. The program shown on the left of Figure 1 is a minor variant of the program in [15, p. 64], computing the floor of the square root of a natural number a. The basic semi-algebraic transition system shown on the right of the figure models the (second loop of the) program. Note that even the original program in [15], which has the disequality $c \neq 1$ in the guard of the second loop, can be modeled as a basic semi-algebraic transition system (by translating $c \neq 1$ as $c \leq 0 \lor c \geq 2$ and then having four transitions instead of two). The variant in Figure 1 has been adopted just for presentation purposes: its analysis leads to the same invariants that are computed when analyzing the original program.

2.3 Abstract Interpretation

Abstract interpretation [11] is a general theory of approximation of the behavior of dynamic discrete systems. One of its classical applications is the inference of invariant properties of transition systems [12]. This is done by specifying the set

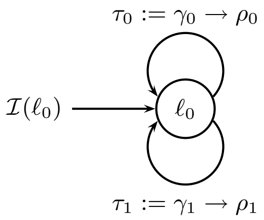
```
{ a ≥ 0 }

b := 0; c := 1;

while c² ≤ a do
    c := 2c;
end while

ℓ₀ : while c ≥ 2 do
        c := c/2;
        if (b + c)² ≤ a then
            b := b + c;
        end if
     end while

     return b;
```



$$\mathcal{I}(\ell_0) := \{a \geq 0, b = 0, c \geq 1, c^2 \geq a + 1\}$$

$$\gamma_0 := \left\{c \geq 2, (2b + c)^2 \leq 4a\right\}$$
$$\rho_0 := (a', b', c') = (a, b + c/2, c/2)$$

$$\gamma_1 := \left\{c \geq 2, (2b + c)^2 \geq 4a + 4\right\}$$
$$\rho_1 := (a', b', c') = (a, b, c/2)$$

Fig. 1. A program and its model as a basic semi-algebraic transition system

of reachable states of the given transition system as the solution of a system of fixpoint equations. The concrete behavior of the transition system is then over-approximated by setting up a corresponding system of equations defined over an *abstract domain*, providing computable representations for the *abstract properties* that are of interest for the analysis, as well as *abstract operations* that are sound approximations of the *concrete operations* used by the transition system being analyzed. An approximation of one solution of the system of abstract equations can be found iteratively, possibly applying further conservative approximations and using convergence acceleration methods, such as *widenings* [11]. One of the main advantages of this methodology for the inference of invariant properties is that the correctness of the obtained results follows by design.

More specifically, given a transition system $S = (\boldsymbol{x}, \mathcal{L}, \mathcal{T}, \mathcal{I})$, the set of its reachable states $\mathrm{reach}(S)$ can be characterized by means of the following system of fixpoint equations where, for each $\ell \in \mathcal{L}$, we have the equation

$$\mathrm{reach}_\ell(S) = \mathcal{I}(\ell) \cup \bigcup \left\{ \rho\big(\mathrm{reach}_{\ell'}(S) \cap \gamma\big) \;\middle|\; (\ell', \ell, \gamma, \rho) \in \mathcal{T} \right\}. \tag{1}$$

The least fixpoint of this system of equations, with respect to the pointwise extension of the subset ordering on $\wp(\mathbb{R}^n)$, is $\mathrm{reach}(S)$; any overapproximation of $\mathrm{reach}(S)$ yields an invariant map for S.

3 Approximating Basic Semi-algebraic Sets

The construction of our abstract domain is analogous to that in [8], where *pseudo-ideals* of polynomials are introduced to infer polynomial *equalities* as invariants, while still reasoning in the framework of linear algebra. Here, we extend this approach so as to handle polynomial *inequalities* as invariants.

In [8], the basic underlying definition is that of a *vector space* of polynomials:

Definition 6. (Vector space.) *A set of polynomials $V \subseteq \mathbb{R}[\boldsymbol{x}]$ is a* vector space *if (1) $0 \in V$; and (2) $\lambda p + \mu q \in V$ whenever $p, q \in V$ and $\lambda, \mu \in \mathbb{R}$. For each $Q \subseteq \mathbb{R}[\boldsymbol{x}]$, the* vector space spanned by Q, *denoted by $\mathcal{V}(Q)$, is the least vector space containing Q, that is,*

$$\mathcal{V}(Q) := \left\{ \sum_{i=1}^{s} \lambda_i q_i \in \mathbb{R}[\boldsymbol{x}] \;\middle|\; s \in \mathbb{N}, \forall i \in \{1, \ldots, s\} : \lambda_i \in \mathbb{R}, q_i \in Q \right\}.$$

Given a vector space V, we associate the constraint $p = 0$ to any $p \in V$. Notice that, if $p, q \in \mathbb{R}[\boldsymbol{x}]$ and $\boldsymbol{v} \in \mathbb{R}^n$ are such that $p(\boldsymbol{v}) = 0$ and $q(\boldsymbol{v}) = 0$, then $(\lambda p + \mu q)(\boldsymbol{v}) = 0$, for any $\lambda, \mu \in \mathbb{R}$. Further, for any $\boldsymbol{v} \in \mathbb{R}^n$, the zero polynomial trivially satisfies $0(\boldsymbol{v}) = 0$. Thus, the set of polynomials that evaluate to 0 on a set of states $S \in \wp(\mathbb{R}^n)$, that is $\{ p \in \mathbb{R}[\boldsymbol{x}] \mid \forall \boldsymbol{v} \in S : p(\boldsymbol{v}) = 0 \}$, has the structure of a vector space. Unfortunately, this vector space has infinite dimension. In order to work with objects of finite dimension, it is necessary to approximate by bounding the degrees of the polynomials.

Moreover, when considering polynomials as elements of a vector space, the algebraic relationships between terms such as x_1, x_2 and $x_1 x_2$ are lost. For instance, consider the vector space $\mathcal{V}\bigl(\{x_1, x_2 - x_1 x_2\}\bigr)$, generated by the polynomial equalities $x_1 = 0$ and $x_2 = x_1 x_2$. Then, even though the polynomial equality $x_2 = 0$ is semantically entailed by the previous ones, $x_2 \notin \mathcal{V}\bigl(\{x_1, x_2 - x_1 x_2\}\bigr)$. The reason is that the vector space generated by x_1 and $x_2 - x_1 x_2$ only includes the *linear* combinations of its generators, whereas in the case above x_2 can only be obtained by a *nonlinear* combination of the generators, namely $x_2 = x_2 \cdot (x_1) + 1 \cdot (x_2 - x_1 x_2)$. This problem can be solved by adding the polynomial $x_1 x_2$ to the set of generators, so that the polynomial $x_2 \in \mathcal{V}\bigl(\{x_1, x_1 x_2, x_2 - x_1 x_2\}\bigr)$ can be obtained by the linear combination $0 \cdot (x_1) + 1 \cdot (x_1 x_2) + 1 \cdot (x_2 - x_1 x_2)$.

In general, in order to reduce the loss of precision due to the linearization of the abstraction, additional polynomials are added taking into account that, when $p \in \mathbb{R}[\boldsymbol{x}]$ and $\boldsymbol{v} \in \mathbb{R}^n$ are such that $p(\boldsymbol{v}) = 0$, we have $(pq)(\boldsymbol{v}) = 0$ for each $q \in \mathbb{R}[\boldsymbol{x}]$. Therefore, pseudo-ideals are defined as follows:

Definition 7. (Pseudo-ideal.) *A* pseudo-ideal of degree $d \in \mathbb{N}$ *is a vector space $P \subseteq \mathbb{R}_d[\boldsymbol{x}]$ such that $pq \in P$ whenever $p \in P$, $q \in \mathbb{R}[\boldsymbol{x}]$ and $\deg(pq) \leq d$. For each $Q \subseteq \mathbb{R}_d[\boldsymbol{x}]$, the* pseudo-ideal of degree d spanned by Q, *denoted by* $\mathrm{pseudo}_d(Q)$, *is the least pseudo-ideal of degree d containing Q.*

Pseudo-ideals are closed under addition, product by scalars and degree-bounded product by polynomials. For instance, for the example above $x_1 x_2 \in \mathrm{pseudo}_2\bigl(\{x_1, x_2\}\bigr)$. Pseudo-ideals are the elements of the abstract domain used in [8].

In order to extend this methodology to the generation of invariant polynomial inequalities, a first, necessary step is the identification, in the basic semi-algebraic context, of an adequate algebraic structure playing the same role of vector spaces for polynomial equalities. It turns out that *polynomial cones* are the right notion:

Definition 8. (Polynomial cone.) *A set of polynomials $C \subseteq \mathbb{R}[\boldsymbol{x}]$ is a* polynomial cone *if* (1) $1 \in C$*; and* (2) $\lambda p + \mu q \in C$ *whenever* $p, q \in C$ *and* $\lambda, \mu \in \mathbb{R}_+$. *For each* $Q \subseteq \mathbb{R}[\boldsymbol{x}]$*, the* polynomial cone generated by Q*, denoted by* $\mathcal{C}(Q)$*, is the least polynomial cone containing* Q*, that is,*

$$\mathcal{C}(Q) := \left\{ \lambda + \sum_{i=1}^{s} \lambda_i q_i \in \mathbb{R}[\boldsymbol{x}] \;\middle|\; \lambda \in \mathbb{R}_+, s \in \mathbb{N}, \forall i \in \{1, \ldots, s\} : \lambda_i \in \mathbb{R}_+, q_i \in Q \right\}.$$

Mimicking the reasoning done before for vector spaces, we associate the constraint $p \geq 0$ to any polynomial p in the polynomial cone C. Consider the basic semi-algebraic set defined by the constraint system

$$\psi = \{f_1 = 0, \ldots, f_h = 0, g_1 \geq 0, \ldots, g_k \geq 0\} \tag{2}$$

where, for each $i = 1, \ldots, h$ and $j = 1, \ldots, k$, we have $f_i, g_j \in \mathbb{R}[\boldsymbol{x}]$. Then, the set of polynomial inequalities that are consequences of ψ define a polynomial cone. Indeed, $\psi \implies (1 \geq 0)$ trivially; and, if $\psi \implies (p \geq 0)$ and $\psi \implies (q \geq 0)$, clearly $\psi \implies (\lambda p + \mu q \geq 0)$ for each $\lambda, \mu \in \mathbb{R}_+$. As was the case for the vector space of polynomials, this set of polynomials has infinite dimension. In order to deal with objects of finite dimension, we again fix an upper bound for the degrees of the polynomials. Moreover, to mitigate the precision loss due to linearization, we close this cone with respect to degree-bounded product by polynomials. Thus, the analog of pseudo-ideals in the basic semi-algebraic setting are *product-closed polynomial cones*:

Definition 9. (Product-closed polynomial cone.) *A* product-closed polynomial cone of degree $d \in \mathbb{N}$ *is a polynomial cone* $C \subseteq \mathbb{R}_d[\boldsymbol{x}]$ *satisfying:*

(1) $pq \in C$ *whenever* $\{p, -p\} \subseteq C$, $q \in \mathbb{R}[\boldsymbol{x}]$ *and* $\deg(pq) \leq d$*;*
(2) $pq \in C$ *whenever* $p, q \in C$ *and* $\deg(pq) \leq d$.

For each $Q \subseteq \mathbb{R}_d[\boldsymbol{x}]$*, the* product-closed polynomial cone of degree d generated by Q*, denoted by* $\mathrm{prod}_d(Q)$*, is the least product-closed polynomial cone of degree* d *containing* Q.

Let ψ be a constraint system defining a basic semi-algebraic set. Then, once the degree bound $d \in \mathbb{N}$ is fixed, ψ can be abstracted by the product-closed polynomial cone $\mathrm{prod}_d\big(\mathrm{poly}(\psi) \cap \mathbb{R}_d[\boldsymbol{x}]\big)$. The approximation forgets those polynomials occurring in ψ having a degree greater than d. Also note that the precision of the approximation depends on the specific constraint system ψ.

The abstraction is clearly sound. For the linear case it is also complete. In fact, consider any finite set of linear constraints $\varphi = \{p_1 \geq 0, \ldots, p_k \geq 0\}$, which we assume to be satisfiable. Then, the corresponding product-closed polynomial cone of degree 1 is $L = \mathrm{prod}_1\big(\mathrm{poly}(\varphi)\big) = \mathcal{C}\big(\mathrm{poly}(\varphi)\big)$, whose elements are the nonnegative linear consequences of φ. On the other hand, if $p \in \mathbb{R}_1[\boldsymbol{x}]$ is a linear polynomial such that $\varphi \implies (p \geq 0)$, then by Farkas' lemma there exists $\boldsymbol{\mu} = (\mu_0, \ldots, \mu_k) \in \mathbb{R}_+^{k+1}$ such that $p = \mu_0 + \sum_{i=1}^{k} \mu_i p_i$; in other words, $p \in L$.

In the general nonlinear setting, the abstraction constituted by product-closed polynomial cones is not complete. Notice however that the set of *all* invariant polynomial inequalities is not computable in basic semi-algebraic transition systems. Worse, the set of all invariant *linear equalities* is not computable in transition systems even if restricted to linear equality guards [24].

3.1 Representation

Given a finitely generated polynomial cone, by exploiting classical duality results [33], each polynomial generator p is interpreted as the constraint $p \geq 0$; these polynomial constraints are then linearized so as to define a convex polyhedron on an extended ambient space. The linearization in the abstraction process implies that all terms are considered as different variables. For instance, in Example 1, the terms a, b, c, c^2 are all regarded as different and potentially independent variables, and the initial condition $\mathcal{I}(\ell_0) = \{a \geq 0, b = 0, c \geq 1, c^2 \geq a + 1\}$ is interpreted as defining, by means of 4 constraints, a convex polyhedron in an ambient space of dimension at least 4. In general, given a transition system on an n-tuple $\boldsymbol{x}$ of variables and a degree bound d, the introduction of the auxiliary variables, standing for all the nonlinear terms of degree at most d, yields an m-tuple $\boldsymbol{y}$ of variables, where each y_i corresponds to one of the $m = \binom{n+d}{d} - 1$ different terms $\boldsymbol{x}^{\boldsymbol{\alpha}} \in \mathbb{R}_d[\boldsymbol{x}]$, where $\boldsymbol{\alpha} \neq \mathbf{0}$. Thus, computation in the abstract domain of cones of degree d is only feasible provided d is small, e.g., 2 or 3. In the following, we will denote each y_i by writing the corresponding term $\boldsymbol{x}^{\boldsymbol{\alpha}}$.

It remains to be seen how the linearized constraint system can be closed, according to Definition 9, with respect to bounded-degree product by polynomials. Rather than trying to obtain the exact product-closed polynomial cone by means of a potentially expensive fixpoint computation, we actually approximate it as follows. Consider the constraint system ψ as defined in (2). Let $\mathcal{M}(g_1, \ldots, g_k)$ be the *multiplicative monoid* generated by the g_j's, i.e., the set of finite products of g_j's including 1 (the empty product). Let us consider the polynomials $p = \sum_{i=1}^{h} r_i f_i + \sum_j \lambda_j g'_j$, where for each $i = 1, \ldots, h$, $r_i \in \mathbb{R}[\boldsymbol{x}]$ is such that $\deg(r_i f_i) \leq d$, and for each j, $\lambda_j \in \mathbb{R}_+$ and $g'_j \in \mathcal{M}(g_1, \ldots, g_k) \cap \mathbb{R}_d[\boldsymbol{x}]$. These polynomials belong to $\mathrm{prod}_d\bigl(\mathrm{poly}(\psi) \cap \mathbb{R}_d[\boldsymbol{x}]\bigr)$, and thus soundly overapproximate the basic semi-algebraic set corresponding to ψ. Algorithm enrich_d, given in Figure 2, computes this approximation, which in practice provides comparable precision to the product closure at much less computational cost.

Example 2. Consider Example 1. The application of procedure enrich_2 to the initial condition $\phi = \mathcal{I}(\ell_0)$ yields the system of constraints

$$\begin{aligned}
\phi' &= \mathrm{enrich}_2\bigl(\phi \cap \mathbb{R}_2[\boldsymbol{x}]\bigr) \\
&= \mathrm{enrich}_2\bigl(\{b = 0\} \cup \{a \geq 0, c \geq 1, c^2 \geq a + 1\}\bigr) \\
&= \{b = 0, ab = 0, b^2 = 0, bc = 0\} \\
&\quad \cup \{a \geq 0, c \geq 1, c^2 \geq a + 1, a^2 \geq 0, c^2 \geq 1, ac \geq 0\}.
\end{aligned}$$

Require: A finite set of polynomial equalities $\varphi = \{f_1 = 0, \ldots, f_h = 0\}$ and a finite set of polynomial inequalities $\psi = \{g_1 \geq 0, \ldots, g_k \geq 0\}$.
Ensure: $\varphi' = \{f'_1 = 0, \ldots, f'_{h'} = 0\}$ and $\psi' = \{g'_1 \geq 0, \ldots, g'_{k'} \geq 0\}$ are finite sets of polynomial equalities and inequalities, respectively, such that $\mathrm{poly}(\varphi' \cup \psi') \subseteq \mathbb{R}_d[\boldsymbol{x}]$ and $\mathcal{C}\big(\mathrm{poly}(\varphi \cup \psi) \cap \mathbb{R}_d[\boldsymbol{x}]\big) \subseteq \mathcal{C}\big(\mathrm{poly}(\varphi' \cup \psi')\big) \subseteq \mathrm{prod}_d\big(\mathrm{poly}(\varphi \cup \psi) \cap \mathbb{R}_d[\boldsymbol{x}]\big)$.

```
φ' := ψ' := ∅
for all (f = 0) ∈ φ do
  if deg(f) ≤ d then
    for all x^α such that deg(x^α) ≤ d − deg(f) do
      φ' := φ' ∪ {x^α f = 0}
for all finite product g' of g's such that (g ≥ 0) ∈ ψ do
  if deg(g') ≤ d then
    ψ' := ψ' ∪ {g' ≥ 0}
```

Fig. 2. Algorithm enrich_d

In this case, $\mathcal{C}\big(\mathrm{poly}(\phi')\big) = \mathrm{prod}_2\big(\mathrm{poly}(\phi) \cap \mathbb{R}_2[\boldsymbol{x}]\big)$. In general, a precision loss may occur; for instance, letting $\chi = \{x \geq 0, x^2 \geq 0, y - y^2 \geq 0, y^2 \geq 0\}$, we have $\chi = \mathrm{enrich}_2(\chi)$, but $(xy \geq 0) \in \mathrm{prod}_2\big(\mathrm{poly}(\chi)\big) \setminus \mathcal{C}\big(\mathrm{poly}(\chi)\big)$.

3.2 Abstract Semantics

In this section we review the operations required in order to perform abstract interpretation of transition systems using polynomial cones as abstract values.

Union. Given two (finitely generated) polynomial cones C_1 and C_2 representing the polynomial constraint systems ψ_1 and ψ_2, respectively, we would like to approximate the union of the corresponding basic semi-algebraic sets using another basic semi-algebraic set. By duality, this amounts to computing the intersection cone $C_1 \cap C_2$: for each $p \in C_1 \cap C_2$ and $\boldsymbol{v} \in \mathbb{R}^n$ such that $\psi_1(\boldsymbol{v}) \vee \psi_2(\boldsymbol{v})$, either $\psi_1(\boldsymbol{v})$, so that $p(\boldsymbol{v}) \geq 0$ as $p \in C_1$; or $\psi_2(\boldsymbol{v})$, so that $p(\boldsymbol{v}) \geq 0$ as $p \in C_2$. Thus, the approximation is sound. At the implementation level, since polynomial cones are represented by means of their (linearized) duals, this intersection of cones corresponds to the convex polyhedral hull operation.

Intersection. Given two (finitely generated) polynomial cones $C_1 = \mathcal{C}(Q_1)$ and $C_2 = \mathcal{C}(Q_2)$, we would like to compute the intersection of the respective basic semi-algebraic sets. Then a sound approximation is to compute the cone spanned by the union of the generators, $\mathcal{C}(Q_1 \cup Q_2)$. In order to reduce the loss of precision due to linearization, we enrich this cone with respect to degree-bounded product by polynomials as explained above. Thus, the polynomial cone corresponding to the intersection is $\mathrm{enrich}_d(Q_1 \cup Q_2)$.

Update. Each basic semi-algebraic update map $\rho \colon \mathbb{R}^n \to \wp(\mathbb{R}^n)$, defined over the original n-tuple of variables $\boldsymbol{x}$, is approximated by a *linearized* update map $\rho^\star \colon \mathbb{R}^m \to \wp(\mathbb{R}^m)$, where $m = \binom{n+d}{d} - 1$, defined over the extended m-tuple $\boldsymbol{y}$ of terms. The new update map $\rho^\star$ is obtained by composing a sequence of simpler maps, each one approximating the effect of ρ on a single term. For the sake of notation, if variable y_i corresponds to term $\boldsymbol{x}^{\boldsymbol{\alpha}}$ and $p \in \mathbb{R}_d[\boldsymbol{x}]$, let $\boldsymbol{x}^{\boldsymbol{\alpha}} \mapsto p$ denote the deterministic update map such that, for each $\boldsymbol{w} \in \mathbb{R}^m$,

$$(\boldsymbol{x}^{\boldsymbol{\alpha}} \mapsto p)(\boldsymbol{w}) := \Big\{ \big(w_1, \dots, w_{i-1}, p(\boldsymbol{w}), w_{i+1}, \dots, w_m\big) \Big\} \subseteq \mathbb{R}^m. \tag{3}$$

Note that the (possibly nonlinear) polynomial $p \in \mathbb{R}_d[\boldsymbol{x}]$ on the original tuple of variables is interpreted as a linear polynomial $p \in \mathbb{R}_1[\boldsymbol{y}]$ on the extended ambient space, so that Equation (3) indeed defines an affine map. Similarly, $\boldsymbol{x}^{\boldsymbol{\alpha}} \mapsto ?$ denotes the nondeterministic update map such that, for each $\boldsymbol{w} \in \mathbb{R}^m$,

$$(\boldsymbol{x}^{\boldsymbol{\alpha}} \mapsto ?)(\boldsymbol{w}) := \big\{ \, (w_1, \dots, w_{i-1}, u, w_{i+1}, \dots, w_m) \in \mathbb{R}^m \bigm| u \in \mathbb{R} \, \big\}.$$

By hypothesis, ρ is defined by $k \leq n$ polynomials $p_1, \dots, p_k \in \mathbb{R}[\boldsymbol{x}]$ and distinct indices $i_1, \dots, i_k \in \{1, \dots, n\}$ such that, for each $\boldsymbol{v} \in \mathbb{R}^n$,

$$\rho(\boldsymbol{v}) = \big\{ \, (v_1', \dots, v_n') \in \mathbb{R}^n \bigm| v_{i_1}' = p_1(\boldsymbol{v}), \dots, v_{i_k}' = p_k(\boldsymbol{v}) \, \big\}.$$

Then, for each term $\boldsymbol{x}^{\boldsymbol{\alpha}} \in \mathbb{R}_d[\boldsymbol{x}]$ where $\boldsymbol{\alpha} \neq \boldsymbol{0}$, we distinguish the following cases:

- Suppose there exists $j \in \{1, \dots, n\}$ such that $\alpha_j > 0$ and $j \notin \{i_1, \dots, i_k\}$. This means that ρ nondeterministically updates at least one of the relevant factors of the term $\boldsymbol{x}^{\boldsymbol{\alpha}}$. Thus, we conservatively approximate the overall effect of ρ on $\boldsymbol{x}^{\boldsymbol{\alpha}}$ using $\boldsymbol{x}^{\boldsymbol{\alpha}} \mapsto ?$, as if it was a nondeterministic assignment.
- Suppose now that, for each $j = 1, \dots, n$, if $\alpha_j > 0$ then $j \in \{i_1, \dots, i_k\}$, i.e., all the relevant factors of $\boldsymbol{x}^{\boldsymbol{\alpha}}$ are deterministically updated by ρ. Then:
  - if the polynomial $p_{\boldsymbol{\alpha}} := \prod \big\{ \, p_h^{\alpha_j}(\boldsymbol{x}) \bigm| j \in \{1, \dots, n\}, \alpha_j > 0, j = i_h \, \big\}$ is such that $p_{\boldsymbol{\alpha}} \in \mathbb{R}_d[\boldsymbol{x}]$, we apply the affine map $\boldsymbol{x}^{\boldsymbol{\alpha}} \mapsto p_{\boldsymbol{\alpha}}$;
  - otherwise, since we cannot represent the effect of ρ on $\boldsymbol{x}^{\boldsymbol{\alpha}}$, we (again) conservatively overapproximate it as $\boldsymbol{x}^{\boldsymbol{\alpha}} \mapsto ?$.

Since ρ updates all terms simultaneously, these maps are ordered topologically according to the dependencies of terms (possibly adding temporary copies of some term variables, which are eliminated at the end).

Example 3. Consider the transitions of Example 1. For the transition τ_0 we have $\rho_0 \equiv (a', b', c') = (a, b, c/2)$. This update is linearized by composing the affine maps $ac \mapsto ac/2$, $bc \mapsto bc/2$, $c^2 \mapsto c^2/4$ and $c \mapsto c/2$, leading to $\rho_0^\star$ defined as

$$\big(a', b', c', ab', ac', bc', (a^2)', (b^2)', (c^2)'\big) = (a, b, c/2, ab, ac/2, bc/2, a^2, b^2, c^2/4).$$

Test for inclusion. The test for inclusion can be conservatively overapproximated by means of the test for inclusion for convex polyhedra.

Widening. Any widening for convex polyhedra, e.g., the standard widening [14] or the more sophisticated widenings proposed in [2, 3], will serve the purpose of guaranteeing termination, with different trade-offs between efficiency and precision.

Example 4. For the transitions of Example 1, using the abstract semantics shown above, we obtain the invariant

$$\begin{aligned}\mathrm{reach}_{\ell_0}(S) \implies \bigl\{&(b+c)^2 \geq a+1, a \geq b^2, b \geq 0, c \geq 1,\\ &a^2 \geq 0, ab \geq 0, ac \geq 0, b^2 \geq bc, bc \geq b, (c-1)^2 \geq 0\bigr\}.\end{aligned}$$

Notice that all the constraints appearing on the second line are in fact redundant. Some of these, such as $a^2 \geq 0$ and $(c-1)^2 \geq 0$, are trivially redundant in themselves. Other ones are made redundant by the constraints appearing on the first line (for instance, $ab \geq 0$ is implied by $a \geq b^2$ and $b \geq 0$). This phenomenon is due to the interaction of the enrich_d procedure, which adds redundant constraints to polynomial cones, with the underlying linear inequalities inference rules, which are treating different terms as independent variables and, as a consequence, are only able to detect and remove some of the redundancies.

The two constraints $(b+c)^2 \geq a+1$ and $a \geq b^2$ in the invariant above are essential in a formal proof of the (partial) correctness of the program in Figure 1. Note that the computed invariant *assumes* that the integer division $c := c/2$ is correctly modeled by rational division. Such an assumption can be validated by other analyses, e.g., by using a domain of numerical powers [21], which could infer that c evaluates to a power of 2 at location ℓ_0. Since on termination $c = 1$ holds, the conjunction of these constraints implies $(b+1)^2 > a \geq b^2$.

4 Experimental Evaluation

The approach described in this paper has been implemented in a prototype analyzer that infers polynomial inequalities of degree not greater than $d = 2$. The prototype, which is based on the *Parma Polyhedra Library* (PPL) [4], first performs a rather standard *linear* relations analysis, then assumes the linear invariants so obtained for the analysis of (possibly) nonlinear invariants described in the previous sections. We have observed that this preliminary linear analysis improves the results in a significant way. In fact: (1) it ensures that we never obtain less information than is achievable with the linear analysis alone; (2) the availability of “trusted” linear invariants increases the precision of the nonlinear analysis considerably; and (3) the time spent in the linear analysis phase is usually recovered in the quadratic analysis phase. The prototype uses the sophisticate widening operator proposed in [2] enhanced with variations of the “widening up to” technique described in [17] and with the “widening with tokens” technique (a form of delayed widening application) described in [3].

Considering that, with the chosen degree bound $d = 2$, we are working on an ambient space that has a dimension which is quadratic in the number of variables of the transition system being analyzed, and considering that polyhedra operations have exponential worst-case complexity, some care has to be taken in order to analyze systems of realistic complexity. In our prototype, we exploit the capability of the PPL concerning the support of time-bounded computations. All polyhedra operations are subject to a timeout (5 seconds of CPU time in the experimentation we are about to report); when a timeout expires, the PPL abandons (without leaking memory) the current computation and gives control back to the analyzer. This situation is handled by the analyzer by using a less precise operation (such as replacing the precise convex polyhedral hull of two polyhedra P_1 and P_2 by the polyhedron obtained by removing, from a system of constraints defining P_1, all constraints that are not satisfied by P_2) or by simplifying the involved polyhedra resorting to a domain of bounded differences. With this technique we are able to obtain results that are generally quite precise in reasonable time (note that the prototype was not coded with speed in mind).

We have run the prototype analyzer on a benchmark suite constituted by all the programs from the FAST suite [5] (`http://www.lsv.ens-cachan.fr/fast/`), programs taken from the StInG suite [29] (`http://www.stanford.edu/~srirams/Software/sting.html`), all square root algorithms in [15], programs from [9, 18, 28, 34], and a program, `array`, written by the authors. From the StInG suite we have only omitted those programs with nondeterministic assignments where the *rvalue* is bounded by linear expressions (like $0 \geq x' \geq x + y$), since they do not fall into the programming model used here.

A summary of the experimental results is presented in Table 1. Besides the program name, its origin and the number of variables, locations and transitions (columns from 1 to 5, respectively), the table indicates: (1) the CPU time, in seconds, taken to compute our *linear* invariants (column 6) and how they compare with the ones computed by StInG (column 7: '+' means ours are better, '−' means ours are worse, '=' means they are equal, '≠' means they are not comparable); and (2) the time taken to generate *quadratic* invariants (column 8) and whether these invariants improve upon (that is, are not implied by) the linear ones, *taking into account both our linear invariants as well as those generated by StInG* (column 9: '✓' means we improve the precision). The measurements were performed on a PC with an Intel® Xeon™ CPU clocked at 1.80 GHz, equipped with 1 GB of RAM and running GNU/Linux. Notice that for about 80% of the locations, our linear invariants are at least as strong as the ones produced by StInG, and that, in fact, for one third ours are stronger. Most importantly, for about half of the programs, the obtained quadratic invariants improve the precision of the linear analysis.

5 Conclusion

We have presented a technique for generating invariant polynomial inequalities of bounded degree. The technique, which is based on the abstract interpretation framework, consists in overapproximating basic semi-algebraic sets by means of convex polyhedra, and can thus take advantage of all the work done in that field (e.g., refined widening operators, devices able to throttle the complexity of the analysis such as restricted classes of polyhedra, ways of partitioning the vector space and so forth). The application of our prototype implementation to a number of benchmark programs shows that the method can produce non-trivial and useful quadratic invariant inequalities in reasonable time, thus proving the fea-

Table 1. A summary of the experimental results

					Linear analysis		Quadratic analysis	
Program name	Origin	n	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{T}\rvert$	CPU time	vs StInG	CPU time	Improves
`array`		4	5	6	0.2	$+\neq\neq\neq+$	79.8	✓
`bakery`	[34]	2	9	24	18.6	$=\cdots=$	0.2	✓
`barber`	FAST	8	1	12	18.7	$-$	2.7	✓
`berkeley`	FAST	4	1	3	0.0	$+$	0.1	✓
`cars`	StInG	7	1	2	18.5	$\neq$	45.9	✓
`centralserver`	FAST	12	1	8	5.4	$+$	193.4	
`consistency`	FAST	11	1	7	2.5	$=$	10.0	
`consprodjava`	FAST	16	1	14	325.6	$+$	601.9	
`consprodjavaN`	FAST	16	1	14	308.0	$+$	611.6	
`cousot05vmcai`	[9]	4	1	1	0.0	$=$	0.1	✓
`csm`	FAST	14	1	13	29.3	$=$	219.5	
`dekker`	FAST	22	1	22	458.4	$=$	1218.1	
`dragon`	FAST	5	1	12	0.5	$-$	1.4	✓
`efm`	FAST	6	1	5	0.1	$=$	0.3	
`rfm05hscc`	[28]	4	1	2	0.1	$\neq$	38.5	
`firefly`	FAST	4	1	8	0.1	$=$	0.2	✓
`fms`	FAST	22	1	20	893.2	$=$	2795.0	
`freire`	[16]	3	1	1	0.0	$-$	6.4	
`futurbus`	FAST	9	1	9	2.8	$+$	23.2	✓
`heap`	StInG	5	1	4	0.1	$\neq$	10.9	
`illinois`	FAST	4	1	9	0.1	$=$	0.3	✓
`kanban`	FAST	16	1	16	60.5	$=$	340.4	
`lamport`	FAST	11	1	9	3.1	$+$	13.4	
`lifo`	StInG	7	1	10	1.4	$+$	14.8	✓
`lift`	FAST	4	1	5	0.1	$=$	22.1	
`mesi`	FAST	4	1	4	0.0	$=$	0.1	✓
`moesi`	FAST	5	1	4	0.1	$-$	0.3	✓
`multipoll`	FAST	18	1	17	116.3	$=$	476.8	
`peterson`	FAST	14	1	12	17.6	$+$	88.5	
`producer-consumer`	FAST	5	1	3	0.1	$=$	15.5	
`readwrit`	FAST	13	1	9	7.7	$=$	2147.3	
`rtp`	FAST	9	1	12	2.6	$=$	8.9	
`see-saw`	StInG	2	1	4	0.0	$-$	5.3	
`sqroot1`	[15]	2	1	1	0.0	$=$	0.0	✓
`sqroot2`	[15]	3	1	8	0.0	$+$	15.6	✓
`sqroot3`	[15]	3	2	6	0.0	$==$	10.3	✓
`sqroot4`	[15]	4	2	6	10.3	$=+$	8.2	✓
`sqroot5`	[8]	4	1	1	0.0	$+$	6.1	✓
`sqroot6`	[18]	5	1	2	0.0	$=$	15.5	✓
`swim-pool`	StInG	9	1	6	1.5	$+$	46.5	
`synapse`	FAST	3	1	3	0.0	$+$	0.0	✓
`ticket2i`	FAST	6	1	6	0.3	$+$	5.8	
`ticket3i`	FAST	8	1	9	9.5	$+$	82.6	
`train-beacon`	StInG	3	4	12	0.1	$=--=$	20.5	
`train-one-loc`	StInG	3	1	6	0.0	$-$	0.4	
`ttp`	FAST	9	4	17	9.3	$++++$	126.9	

sibility of the automatic inference of nonlinear invariant inequalities (something that was previously unclear).

For future work, we want to generalize our definition of basic semi-algebraic transition system so as to capture a form of nondeterministic assignments where the *rvalue* is bounded by means of polynomial inequalities, rather than being completely unknown. We would also like to increase the precision of the approach by incorporating, in the enrich_d algorithm, other forms of inference, such as *relational arithmetic* [1, 32]. This technique allows to infer constraints on the qualitative relationship of an expression to its arguments and can be expressed by a number of axiom schemata such as $(x > 0 \wedge y > 0) \implies \bigl(x \bowtie 1 \implies xy \bowtie y\bigr)$, which is valid for each $\bowtie \in \{=, \neq, \leq, <, \geq, >\}$. Finally, there is much room for improving the prototype implementation. To start with, we believe its performance can be greatly enhanced (there are a number of well-known techniques that we are not currently using); this may even bring us to the successful inference of cubic invariants for simple programs. The simplification of the analysis results is another natural candidate for this line of work.

Acknowledgments. The authors are grateful to Deepak Kapur and Alessandro Zaccagnini for their help and comments.

References

1. R. Bagnara. *Data-Flow Analysis for Constraint Logic-Based Languages.* PhD thesis, Dipartimento di Informatica, Università di Pisa, Pisa, Italy, March 1997.
2. R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. In *Proc. SAS 2003*, vol. 2694 of *LNCS*, pp. 337–354, San Diego.
3. R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. *Science of Computer Programming*, 2005. To appear.
4. R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In *Proc. SAS 2002*, vol. 2477 of *LNCS*, pp. 213–229, Madrid, Spain.
5. S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: Fast Acceleration of Symbolic Transition systems. In *Proc. CAV 2003*, vol. 2725 of *LNCS*, pp. 118–121, Boulder, CO.
6. S. Bensalem, M. Bozga, J.-C. Fernandez, L. Ghirvu, and Y. Lakhnech. A transformational approach for generating non-linear invariants. In *Proc. SAS 2000*, vol. 1824 of *LNCS*, pp. 58–74, Santa Barbara, CA.
7. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *Proc. PLDI 2003*, pp. 196–207, San Diego, CA.
8. M. Colón. Approximating the algebraic relational semantics of imperative programs. In *Proc. SAS 2004*, vol. 3148 of *LNCS*, pp. 296–311, Verona, Italy.
9. P. Cousot. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In *Proc. VMCAI 2005*, vol. 3385 of *LNCS*, pp. 1–24, Paris, France.
10. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. ISOP 1976*, pp. 106–130, Paris, France.

11. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. POPL 1977*, pp. 238–252, New York.
12. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. POPL 1979*, pp. 269–282, New York.
13. P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTRÉE analyzer. In *Proc. ESOP 2005*, vol. 3444 of *LNCS*, pp. 21–30, Edinburgh, UK.
14. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. POPL 1978*, pp. 84–96, Tucson, AR.
15. E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.
16. P. Freire. SQRT. Retrieved April 10, 2005, from `http://www.pedrofreire.com/sqrt`, 2002.
17. N. Halbwachs. Delay analysis in synchronous programs. In *Proc. CAV 1993*, vol. 697 of *LNCS*, pp. 333–346, Elounda, Greece.
18. P. Hsieh. How to calculate square roots. Retrieved April 10, 2005, from `http://www.azillionmonkeys.com/qed/sqroot.html`, 2004.
19. D. Kapur. Automatically generating loop invariants using quantifier elimination. In *Proc. ACA 2004*, Beaumont, Texas.
20. M. Karr. Affine relationships among variables of a program. *Acta Informatica*, 6:133–151, 1976.
21. I. Mastroeni. Algebraic power analysis by abstract interpretation. *Higher-Order and Symbolic Computation*, 17(4):297–345, 2004.
22. A. Miné. The octagon abstract domain. In *Proc. WCRE'01*, pp. 310–319, Stuttgart, Germany.
23. M. Müller-Olm and H. Seidl. Computing polynomial program invariants. *Information Processing Letters*, 91(5):233–244, 2004.
24. M. Müller-Olm and H. Seidl. A note on Karr's algorithm. In *Proc. ICALP 2004*, vol. 3142 of *LNCS*, pp. 1016–1028, Turku, Finland.
25. M. Müller-Olm and H. Seidl. Precise interprocedural analysis through linear algebra. In *Proc. POPL 2004*, pp. 330–341, Venice, Italy.
26. E. Rodríguez-Carbonell and D. Kapur. An abstract interpretation approach for automatic generation of polynomial invariants. In *Proc. SAS 2004*, vol. 3148 of *LNCS*, pp. 280–295, Verona, Italy.
27. E. Rodríguez-Carbonell and D. Kapur. Automatic generation of polynomial loop invariants: Algebraic foundations. In *Proc. ISSAC 2004*, pp. 266–273, Santander.
28. M. Roozbehani, E. Feron, and A. Megrestki. Modeling, optimization and computation for software verification. In *Proc. HSCC 2005*, pp. 606–622, Zürich.
29. S. Sankaranarayanan, H. Sipma, and Z. Manna. Constraint-based linear-relations analysis. In *Proc. SAS 2004*, vol. 3148 of *LNCS*, pp. 53–68, Verona, Italy.
30. S. Sankaranarayanan, H. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *Proc. VMCAI 2005*, vol. 3385 of *LNCS*, pp. 25–41, Paris, France.
31. S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Non-linear loop invariant generation using Gröbner bases. In *Proc. POPL 2004*, pp. 318–329, Venice, Italy.
32. R. Simmons. Commonsense arithmetic reasoning. In *Proc. AAAI 1986*, vol. 1, pp. 118–124, Philadelphia, PA.
33. J. Stoer and C. Witzgall. *Convexity and Optimization in Finite Dimensions I.* Springer-Verlag, Berlin, 1970.
34. A. Tiwari, H. Rueß, H. Saïdi, and N. Shankar. A technique for invariant generation. In *Proc. TACAS 2001*, vol. 2031 of *LNCS*, pp. 113–127, Genova, Italy.