Linear Invariant Generation Using Non-Linear Constraint Solving

Michael A. Colón, Sriram Sankaranarayanan and Henny B. Sipma ⋆

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

Abstract. We present a new method for the generation of linear invariants which reduces the problem to a non-linear constraint solving problem. Our method, based on *Farkas' Lemma*, synthesizes linear invariants by extracting non-linear constraints on the coefficients of a target invariant from a program. These constraints guarantee that the linear invariant is inductive. We then apply existing techniques, including specialized quantifier elimination methods over the reals, to solve these non-linear constraints. Our method has the advantage of being complete for inductive invariants. To our knowledge, this is the first sound and complete technique for generating inductive invariants of this form. We illustrate the practicality of our method on several examples, including cases in which traditional methods based on abstract interpretation with widening fail to generate sufficiently strong invariants.

1 Introduction

An *invariant* assertion of a program at a location is an assertion over the program variables that remains true whenever the location is reached. Invariants are essential for verifying the correctness of programs. The automatic generation of invariants is useful both as a direct means of checking program specifications and as an indirect means of obtaining *intermediate assertions* that can be used as lemmas for proving other safety and liveness properties [16].

An assertion is said to be *inductive* at a program location if it holds the first time the location is reached and is preserved under every cycle back to the location. It has been established that all inductive assertions are invariant. Furthermore, the standard method for proving a given assertion invariant is to find an inductive assertion that strengthens it [16]. Thus invariant generation methods are, normally, methods for generating inductive assertions.

The dominant invariant generation technique is *abstract interpretation* [6]. The main idea behind this approach is to perform an approximate symbolic

⋆ This research was supported in part by NSF(ITR) grant CCR-01-21403, by NSF grant CCR-99-00984-001, by ARO grant DAAD19-01-1-0723, and by ARPA/AF contracts F33615-00-C-1693 and F33615-99-C-3014.

execution of the program until an assertion is reached that remains unchanged by further executions of the program. This assertion can be shown to be inductive, and hence invariant. However, in order to guarantee termination, the method introduces imprecision by use of an extrapolation operator called *widening*. This operator often causes the technique to produce weak invariants. The design of a widening operator with some guarantee of completeness remains a key challenge for abstract interpretation based techniques [7, 1]. In fact, tools like HyTech [12] have given up widening in favor of extrapolation heuristics with no convergence guarantees and have reported good results.

In this paper, we generate linear invariants for linear transition systems, a class of programs that is widely studied [7, 1, 11]. A large number of reactive systems may be modeled directly or approximately as linear transition systems [13]. Rather than perform a least fixed point computation by iteration, we solve constraints on the coefficients $c_1, \ldots, c_n, d$ of a target invariant $c_1x_1 + \cdots + c_nx_n + d \leq 0$. The constraints encode the conditions for inductiveness of the inequality. The solution method is exact, thus guaranteeing that all inductive invariants of this form can be found. The main advantage is that there is no heuristic widening operation, and thus the method does not suffer from the problem of overshooting invariants. On the other hand, the method generates *non-linear* constraints that can be solved only for small or medium size problem instances using current techniques. This disadvantage notwithstanding, we demonstrate our approach on some examples drawn from the literature. We show that in many cases, our method generates invariants that forward propagation with widening misses. Non-linear constraint solving is an active area of research, and our approach will become increasingly practical as more effective techniques for solving these constraints are developed.

The rest of the paper is organized as follows: Section 2 presents some preliminary definitions. In Section 3, we present a method for generating constraints using Farkas' Lemma. Techniques for solving non-linear constraints are briefly described in Section 4. Section 5 illustrates the method on several examples, and finally, Section 6 concludes with a discussion of the advantages and drawbacks of the approach.

2 Preliminaries

In this section, we provide some key definitions and present Farkas' Lemma, the basis of our approach.

Transition Systems and Invariants

Definition 1 (Transition System) A *transition system* $P : \langle V, L, l_0, \Theta, \mathcal{T} \rangle$ consists of a set of *variables* V, a set of *locations* L, an *initial location* l_0, an *initial assertion* Θ over the variables V, and a set of *transitions* $\mathcal{T}$. Each transition $\tau \in \mathcal{T}$ is a tuple $\langle l, l', \rho_\tau \rangle$, where $l, l' \in L$ are the *pre* and *post locations*, and ρ_τ is the *transition relation*, an assertion over $V \cup V'$, where V represents current-state variables and its primed version V' represents the next-state variables.

Throughout the paper, unless otherwise stated, we assume that the set of variables $V = \{x_1, \ldots, x_n\}$ is fixed. Furthermore, given an assertion ψ over the variables V of a transition system, ψ' denotes the assertion obtained by replacing each variable $x \in V$ by $x' \in V'$.

The *control-flow graph* (CFG) of a transition system is a graph whose vertices are the locations and whose edges are the transitions. A *path* π of the transition system is a path through its CFG, and the relation ρ_π associated with the path is the composition of the corresponding transition relations.

A *cutset* C of a transition system P is a subset of the locations of P with the property that every cyclic path in P passes through some location in C. A location inside a cutset is called a *cutpoint*. A *basic* path π between two cutpoints l_i and l_j is a simple path that does not go through any cutpoint, other than the end points.

Definition 2 (Inductive Assertion Map) Given a program P with a cutset C and an assertion $\eta_c(l)$, for each cutpoint l, we say that η_c is an *inductive assertion map* for C if it satisfies the following conditions for all cutpoints l, l':

Initiation For each basic path π from l_0 to l, $\Theta \wedge \rho_\pi \models \eta_c(l)'$.
Consecution For each basic path π from l to l', $\eta_c(l) \wedge \rho_\pi \models \eta_c(l')'$.

Any such partial map may be extended in a natural way to a total map, provided a computable *post image* operator is available. If the cutset C is obvious from the context, we drop the subscript from the map.

Linear and Non-Linear Constraints

A *linear constraint* over V is an inequality of the form $a_1x_1 + \cdots + a_nx_n + b \leq 0$, where $a_1, \cdots, a_n, b$ denote known real-valued coefficients. The constraint is said to be *homogeneous* if $b = 0$ and *inhomogeneous* otherwise. A *linear assertion* over a set of variables V is a conjunction of linear constraints over V. A linear assertion consisting of k inequalities is said to be *k-linear*. Geometrically, the set of points satisfying a linear assertion forms a *polyhedron*. Linear assertions have been thoroughly studied; problems like satisfiability and projection are known to be decidable [17].

Given a set of vectors S, the *cone* generated by S, denoted by *cone*(S), is the set of all the vectors of the form $\lambda_1 s_1 + \cdots + \lambda_m s_m$, where $s_1, \ldots, s_m \in S$ and $\lambda_1, \ldots, \lambda_m \geq 0$. A cone is said to be *finitely generated* if it is *cone*(S) for some finite S. *Polyhedral cones* are cones that can be characterized as the solution spaces of homogeneous linear assertions. A fundamental result states that a cone is finitely generated iff it is polyhedral [17].

A non-linear constraint is an inequality of the form $P \leq 0$, where P is a polynomial on $x_1, \ldots, x_n$. The degree of the constraint is defined as the degree of the polynomial P. A *non-linear assertion* is a conjunction of non-linear constraints. A polynomial P is said to be *reducible* if $P = P_1P_2$, where P_1 and P_2 are polynomials of strictly lower degree than P. The set of points satisfying

$$
\begin{array}{ll}
\textbf{integer } i,j \textbf{ where } i=2\wedge j=0 & L=\{l_0,l_1\},\ V=\{i,j\}, \\
l_0:\textbf{while true do} & \Theta:\ (i=2\wedge j=0),\ \mathcal{T}=\{\tau_0,\tau_1,\tau_2\}, \\
\left[\begin{array}{ll} & i:=i+4 \\ l_1: & \quad\textbf{or} \\ & (i,j):=(i+2,j+1)\end{array}\right] & \begin{array}{l}\tau_0:\ \langle l_0,l_1,true\rangle \\ \tau_1:\ \langle l_1,\ l_0,\ (i'=i+4\wedge j'=j)\rangle \\ \tau_2:\ \langle l_1,\ l_0,\ (i'=i+2\wedge j'=j+1)\rangle\end{array}
\end{array}
$$

Fig. 1. A simple program fragment and the corresponding transition system

a non-linear assertion is called a *semi-algebraic* set. Problems such as satisfiability, projection, intersection and union, though still decidable, have a higher complexity than for linear constraints [2].

We say that a transition system is *linear* if its initial assertion Θ is linear over V and its transition relations ρ_τ are linear assertions over $V \cup V'$. Consequently, for every simple path π, the relation ρ_π is a linear assertion. The rest of this paper deals with linear transition systems. Corresponding to an inductive assertion map, a *k-linear inductive assertion map* ($k > 0$) is defined to be an assertion map, wherein each assertion is k-linear.

Example. Figure 1 shows a simple program fragment over the variables i and j, taken from [7]. As a transition system, it has a two locations l_0 and l_1, and three transitions τ_0, τ_1 and τ_2.

Farkas' Lemma

Farkas' lemma provides a sound and complete method for reasoning about systems of linear inequalities.

Theorem 1 (Farkas' Lemma). *Consider the following system of linear inequalities over real-valued variables $x_1, \ldots, x_n$,*

$$
S:\ \begin{bmatrix} a_{11}x_1 + \cdots + a_{1n}x_n + b_1 \le 0 \\ \vdots \qquad\qquad\qquad \vdots \quad \vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n + b_m \le 0 \end{bmatrix}
$$

When S is satisfiable, it entails a given linear inequality

$$
\psi:\ c_1x_1 + \cdots + c_nx_n + d \le 0
$$

if and only if there exist non-negative real numbers $\lambda_0, \lambda_1, \ldots, \lambda_m$, such that

$$
c_1 = \sum_{i=1}^{m} \lambda_i a_{i1}, \quad \ldots \quad , c_n = \sum_{i=1}^{m} \lambda_i a_{in},\ d = (\sum_{i=1}^{m} \lambda_i b_i) - \lambda_0
$$

Furthermore, S is unsatisfiable if and only if the inequality $1 \le 0$ can be derived as shown above.

We represent applications of the lemma using a tabular notation:

$$
\begin{array}{c|rcrrl}
\lambda_0 & & & & -1 \le 0 & \\
\lambda_1 & a_{11}x_1 + & \cdots & + \; a_{1n}x_n + & b_1 \le 0 & \multirow{3}{*}{$\left.\vphantom{\begin{array}{c}a\\ \vdots\\ a\end{array}}\right\} S$} \\
\vdots & \vdots & & \vdots & \vdots & \\
\lambda_m & a_{m1}x_1 + & \cdots & + \; a_{mn}x_n + & b_m \le 0 & \\
\hline
& c_1x_1 + & \cdots & + \quad c_nx_n + & d \le 0 & \leftarrow \psi \\
& & & & 1 \le 0 & \leftarrow \mathit{false}
\end{array}
$$

The antecedents are placed above the line and the consequences below. For each column, the sum of the column entries above the line, with the appropriate multipliers, must be equal to the entry below the line. If a row corresponds to an inequality, the corresponding multiplier is required to be non-negative. This requirement is dropped for rows corresponding to equalities.

3 Generating Invariants

We now present our approach to linear invariant generation. We represent the invariant

$$c_1x_1 + \cdots c_nx_n + d \le 0$$

in terms of its coefficients $c_1, \ldots, c_n, d$. The main idea behind our technique is to treat these coefficients as unknowns and generate constraints on the coefficients such that any solution corresponds to an inductive assertion. The key to this approach is *Farkas' Lemma*.

Given a transition system and a cutset, we generate a (partial) inductive assertion map η over the cutpoints by encoding initiation and consecution. Let $\eta(l)$ be represented by the assertion $c_{l1}x_1 + \cdots + c_{ln}x_n + d_l \le 0$, where each c_{li} and each d_l is an unknown.[1] The two conditions for the map to be inductive are encoded as follows:

Initiation: For each cutpoint l and each basic path π from l_0 to l, the path may be an *enabled* path, in which case $\Theta \wedge \rho_\pi$ is satisfiable, or the path may be *disabled*, in which case, $\Theta \wedge \rho_\pi$ is unsatisfiable. Initiation can thus be represented by the following table,

$$
\begin{array}{c|rcrrcrrl}
\lambda_0 & & & & & & & -1 \le 0 & \\
\lambda_1 & a_{11}x_1 + & \cdots & + \; a_{1n}x_n & & & & + \quad b_1 \le 0 & \\
\vdots & \vdots & & \vdots & & & & \vdots & \Big\} \; \Theta \\
\lambda_{j-1} & a_{j-1,1}x_1 + & \cdots & + \; a_{j-1,n}x_n & & & & + \; b_{j-1} \le 0 & \\
\lambda_j & a_{j1}x_1 + & \cdots & + \; a_{jn}x_n + & a'_{j1}x'_1 + & \cdots & + \; a'_{jn}x'_n + & b_j \le 0 & \\
\vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & \Big\} \; \rho_\pi \\
\lambda_m & a_{m1}x_1 + & \cdots & + \; a_{mn}x_n + & a'_{m1}x'_1 + & \cdots & + \; a'_{mn}x'_n + & b_m \le 0 & \\
\hline
& & & & c_{l1}x'_1 + & \cdots & + \; c_{ln}x'_n + & d_l \le 0 & \leftarrow \eta(l)' \\
& & & & & & & 1 \le 0 & \leftarrow \mathit{disabled}
\end{array}
$$

[1] We use c, d with subscripts to denote unknowns and a, b with subscripts to denote known values.

where $\lambda_0, \ldots, \lambda_m \geq 0$.

Consecution: For each basic path π from a cutpoint l_i to a cutpoint l_j, we encode the consecution condition, $\eta(l_i) \wedge \rho_\pi \models \eta(l_j)'$, using Farkas' Lemma. Again there are two cases to consider, one when the path is enabled and the other when it is disabled. The constraints are represented by the table shown below:

$$
\begin{array}{r|l}
\mu & c_{l_i1}x_1 + \cdots + c_{l_in}x_n \qquad\qquad\qquad\qquad\qquad\qquad + d_{l_i} \leq 0 \leftarrow \eta(l_i) \\
\lambda_0 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad -1 \leq 0 \\
\lambda_1 & a_{11}x_1 + \cdots + a_{1n}x_n + a'_{11}x'_1 + \cdots + a'_{1n}x'_n + b_1 \leq 0 \\
\vdots & \quad \vdots \qquad\qquad \vdots \qquad\quad \vdots \qquad\qquad\quad \vdots \qquad \vdots \qquad\qquad \rho_\pi \\
\lambda_m & a_{m1}x_1 + \cdots + a_{m1}x_n + a'_{m1}x'_1 + \cdots + a'_{mn}x'_n + b_m \leq 0 \\
\hline
 & \qquad\qquad\qquad\quad c_{l_j1}x'_1 + \cdots + c_{l_jn}x'_n + d_{l_j} \leq 0, \leftarrow \eta(l_j)' \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 1 \leq 0 \leftarrow \mathit{disabled}
\end{array}
$$

where $\mu, \lambda_0, \ldots, \lambda_m \geq 0$.

The constraints corresponding to initiation are linear, as are the constraints corresponding to the disabled case of consecution. However, the constraints for the enabled case of consecution are non-linear due to the presence of the multiplier μ in a row containing unknown coefficients. Methods for solving these constraints are discussed in Section 4.

Example. Consider the program shown in Figure 1. All cycles of the program are cut by l_0. Let $\varphi: \ c_1 i + c_2 j + d \leq 0$ be the target invariant at this cutpoint. The initial condition, $i = 2 \wedge j = 0$, generates the constraints

$$
\begin{array}{r|l}
\lambda_0 & \qquad\qquad\quad -1 \leq 0 \\
\lambda_1 & \ i \qquad\qquad\ -2 = 0 \\
\lambda_2 & \qquad\ j \qquad\quad\ \ = 0 \\
\hline
 & c_1 i + c_2 j + \ d \leq 0 \leftarrow \varphi \\
 & \qquad\qquad\quad\ \ 1 \leq 0 \leftarrow \mathit{disabled}
\end{array}
$$

We can ignore the disabled case since the initial condition is satisfiable. With the requirement that λ_0 be non-negative, we obtain the following constraints:

$$
\psi_\Theta : \ (\exists \lambda_0, \lambda_1, \lambda_2) \begin{bmatrix} c_1 = \lambda_1 \ \wedge \ c_2 = \lambda_2 \ \wedge \\ d = -2\lambda_1 - \lambda_0 \ \wedge \ \lambda_0 \geq 0 \end{bmatrix} \tag{1}
$$

Two paths cycle back to l_0: path π_1 using τ_1 and π_2 through τ_2. For the path π_1, the transition relation ρ_{π_1} is given by $i - i' + 4 = 0 \wedge j - j' = 0$. The two constraints for consecution are given by the table

$$
\begin{array}{r|l}
\mu & c_1 i + c_2 j \qquad\qquad\qquad\quad + d \leq 0 \\
\lambda_0 & \qquad\qquad\qquad\qquad\qquad\ -1 \leq 0 \\
\lambda_1 & \ i \qquad - \quad i' \qquad\qquad + 4 = 0 \\
\lambda_2 & \qquad j \qquad - \quad j' \qquad\quad = 0 \\
\hline
 & \qquad\qquad c_1 i' + c_2 j' + d \leq 0 \leftarrow \varphi \\
 & \qquad\qquad\qquad\qquad\quad\ 1 \leq 0 \leftarrow \mathit{disabled}
\end{array}
$$

resulting in the constraints:

$$
(\exists \overline{\lambda}, \mu) \begin{bmatrix} 0 = \mu c_1 + \lambda_1 \ \wedge \\ 0 = \mu c_2 + \lambda_2 \ \wedge \\ c_1 = -\lambda_1 \ \wedge \ c_2 = -\lambda_2 \ \wedge \\ d = \mu d + 4\lambda_1 - \lambda_0 \ \wedge \\ \mu, \lambda_0 \geq 0 \end{bmatrix} \vee (\exists \overline{\lambda}, \mu) \begin{bmatrix} 0 = \mu c_1 + \lambda_1 \ \wedge \\ 0 = \mu c_2 + \lambda_2 \ \wedge \\ 0 = -\lambda_1 \ \wedge \ 0 = -\lambda_2 \ \wedge \\ 1 = \mu d + 4\lambda_1 - \lambda_0 \ \wedge \\ \mu, \lambda_0 \geq 0 \end{bmatrix} \quad (2)
$$

Similarly, the constraints corresponding to consecution for π_2 are

$$
(\exists \overline{\lambda}, \mu) \begin{bmatrix} 0 = \mu c_1 + \lambda_1 \ \wedge \\ 0 = \mu c_2 + \lambda_2 \ \wedge \\ c_1 = -\lambda_1 \ \wedge \ c_2 = -\lambda_2 \ \wedge \\ d = \mu d + 2\lambda_1 + \lambda_2 - \lambda_0 \wedge \\ \mu, \lambda_0 \geq 0 \end{bmatrix} \vee (\exists \overline{\lambda}, \mu) \begin{bmatrix} 0 = \mu c_1 + \lambda_1 \ \wedge \\ 0 = \mu c_2 + \lambda_2 \ \wedge \\ 0 = -\lambda_1 \ \wedge \ 0 = -\lambda_2 \ \wedge \\ 1 = \mu d + 2\lambda_1 + \lambda_2 - \lambda_0 \wedge \\ \mu, \lambda_0 \geq 0 \end{bmatrix} \quad (3)
$$

We will eliminate the quantifiers and complete the example in Section 4. The technique presented in this section can be easily extended to k-linear assertion maps. Farkas' Lemma may be used in this case to obtain constraints for initiation and consecution with few changes.

Theorem 2. *For any $k > 0$, a $k-$linear assertion map η is inductive iff it is a solution to the system of constraints generated by our method.*

The theorem follows directly from Farkas' Lemma and our constraint generation technique. It states that our technique is sound. Furthermore, completeness holds for those linear invariants that can be proved using linear inductive assertions. A linear program may have a reachable state-space that is not convex, but inductive linear assertions can only characterize convex sets. Hence, there are linear programs which satisfy linear invariants that can only be established by resorting to non-linear or non-convex inductive assertions. This result is demonstrated by the work of Clarke [3].

In theory, having a conjunction of $k > 1$ linear inequalities at each cutpoint is better than a single inequality at each cutpoint. In practice however, we find that the constraints obtained for $k > 1$ are too complex to solve exactly for all but the smallest of systems. Nevertheless, as Section 5 illustrates, our technique with exact solution is powerful even when restricted to 1-linear assertions. Furthermore, k-linear inductive assertions can be approximated iteratively, by starting with 1-linear assertions and strengthening each transition relation with the invariants computed in the previous stage.

4 Solving Constraints

Our method extracts linear and non-linear constraints characterizing inductive assertions. Any solution to these constraints is thus inductive. These constraints may be solved by eliminating the quantified variables. In practice, however, quantifier elimination is a costly process with exponential time complexity. Therefore, we exploit various techniques like factorization and root finding to simplify the constraints, thereby reducing the size of each quantifier elimination instance and that of the result after elimination. In some cases, we are able to generate all the solutions solely by making use of these simplification techniques, without resorting to quantifier elimination. We present some of these techniques and then apply them to solve the constraints generated for our running example. The reader is referred to the comprehensive survey by Bockmayr and Weispfenning [2] on constraint solving for more details.

Linear Constraints

Geometrically, the set of points satisfying an inhomogeneous linear assertion forms a *polyhedron*, and the set satisfying a homogeneous linear assertion is a *polyhedral cone*; elimination of variables corresponds to *projection*. One approach to projection is to compute the generators of the polyhedron and then project these generators on to the free variables. These generators can be computed by the *simplex* method [17] or the *double description* method [9]. In our examples, presented in the next section, we use an implementation of the double description method called PolKa [10].

Alternatively, projection and the computation of generators can be achieved through a quantifier elimination method called *Fourier's* elimination, which eliminates variables from the system of constraints incrementally [2]. Due to its simplicity, Fourier's elimination has been used widely to solve linear constraints, even though its complexity is exponential.

Non-linear Constraints

Non-linear constraints can be solved by direct quantifier elimination or indirect methods using techniques such as factorization and polynomial root solving.

The original breakthrough in quantifier elimination was due to Tarski [18]. However, it was not computationally feasible until Collins introduced *Cylindrical Algebraic Decomposition* [5]. Recently, there have been many practical implementations based on this technique. Notable among them is QEPCAD, which incorporates many improvements to the original CAD algorithm [4]. The time complexity of the algorithm is exponential in the size of the formula. However, the running time can be brought down significantly for low degree polynomials using the elimination at test points method due to Weispfenning [19]. After quantifier elimination, simplification is carried out using factorization and Gröbner Bases. This method has been implemented in REDLOG [8].

Another approach is to use linear programming and delay processing non-linear constraints until they can be linearized or simplified to an extent that they can be solved directly. This approach, which is at the heart of many CLP based solvers [15], works for a surprisingly large variety of problems. However, there are problems that require non-linear constraint solving. Thus, non-linear constraint solvers have been incorporated into the CLP paradigm. For instance, the RISC-CLP(R) system uses quantifier elimination in the background to solve constraints [14].

Heuristics

The constraints we obtain are of a low degree and hence, REDLOG is the most suitable tool. A disadvantage of using quantifier elimination is the size of the result after elimination. In general, the final result after elimination is a boolean combination of, mostly unfactorized, polynomial equalities and inequalities, containing redundant non-linear inequalities that need to be detected and removed. It has been our experience so far that the majority of these reduce to linear factors, and that non-linear irreducible polynomials are rare.

If the polynomials in the result are linear, then the cone containing all the solutions is polyhedral, and a minimal set of generators for this cone can be computed using the double description method. The set of inductive invariants can be completely characterized by considering each of these generators as a constraint.

If there are irredundant and unfactorizable non-linear constraints, the cone of consequences may or may not be finitely generated. However, we are unaware of an efficient method for deciding which case holds. When the cone is not finitely generated, it is not possible to characterize all invariants without using a parametric representation.

We have found a few heuristics that are effective in dealing with non-linear factors: If a polynomial P is reducible, it is always possible to reduce the degree of the constraints by splitting. For instance, the constraint $P_1 P_2 \leq 0$ is equivalent to the disjunction $(P_1 \leq 0 \wedge P_2 \geq 0) \vee (P_1 \geq 0 \wedge P_2 \leq 0)$. Further information about polynomial factorization can be found in standard textbooks on the topic [21].

Another heuristic, especially effective when the result of the elimination is too large to be factorized or simplified, is to set some of the coefficients to zero, in effect restricting the target invariants to those involving only a subset of the variables of the program. Furthermore, since any two-dimensional cone is finitely generated, setting sufficiently many variables to zero always yields a polyhedral cone.

Example. We can now complete our example from the previous section by applying the techniques mentioned in this section to solve the constraints.

The initiation constraint shown in (1) is linear and simplifies to $2c_1 + d \leq 0$. The constraints corresponding to consecution for path π_1 are shown in (2). It is possible to solve them using quantifier elimination. However, a simple substitution on the first clause yields $-\mu\lambda_1 + \lambda_1 = 0 \ \wedge \ -\mu\lambda_2 + \lambda_2 = 0$, which can be simplified to $\mu = 1$ or $\lambda_1 = \lambda_2 = 0$. Branching on both possibilities, we obtain $c_1 \leq 0$ for the former and $c_1 = c_2 = 0, d \leq 0$ for the latter. The other clause can be similarly solved using substitution, yielding $c_1 = c_2 = 0 \wedge d \geq 0$. Consecution

```
integer i, j, k where i = 1 ∧ j = 1 ∧ 0 ≤ k ≤ 1
l0 : while true do
  l1 : (i, j, k) := (i + 1, j + k, k − 1)
```

Fig. 2. INCREMENT

for path π_2 can also be solved using factorization. The final result is:

$$(2c_1 + d \leq 0) \ \wedge \ \begin{bmatrix} (c_1 = c_2 = 0 \wedge d \leq 0) \vee \\ (c_1 \leq 0) \vee \\ (c_1 = c_2 = 0 \wedge d \geq 0) \end{bmatrix} \wedge \begin{bmatrix} (c_1 = c_2 = 0 \wedge d \leq 0) \vee \\ (2c_1 + c_2 \leq 0) \vee \\ (c_1 = c_2 = 0 \wedge d \geq 0) \end{bmatrix}$$

After converting this into DNF and eliminating unsatisfiable disjuncts, we obtain the following generators shown along with the corresponding invariants:

c_1	c_2	d	$c_1 i + c_2 j + d \leq 0$
0	0	−1	$-1 \leq 0$
0	−1	0	$-j \leq 0$
−1	2	2	$-i + 2j + 2 \leq 0$

These match the invariants obtained in [7]. Additional inductive assertions, such as $-i + 2 \leq 0$ can be obtained as consequences of the assertions shown above.

5 Applications

We now demonstrate our approach on several examples.

Increment

Consider the program INCREMENT shown in Figure 2. With each iteration it increments i, while manipulating the variables j and k. Surprisingly, the analysis of Cousot and Halbwachs [7] misses the obvious invariant $i \geq 1$. On the other hand, our method produces the invariants $1 \leq i + k \leq 2$ and $i \geq 1$ in one iteration. This phenomenon, in which the presence of additional, independent variables weakens the invariants generated using abstract interpretation, can be observed in more realistic programs, e.g. the implementation of MERGESORT presented by Wirth [20].

Heapsort

We applied our method to HEAPSORT, shown in Figure 3 and taken from [7]. Arrays and operations involving arrays were not modeled in the transition system, and branches involving array conditions were treated as non-deterministic choices. All cycles are cut by l_3. There are eight paths that go from l_3 back to itself. Upon simplification, the formula obtained after elimination does not contain any non-linear constraint, and the following invariants were easily extracted:

```
integer n, l, r, i, j where l = n/2 + 1 ∧ n ≥ 2 ∧ r = n
realarray T[1...n];
real k;
l0 : if l ≥ 2 then
    l0^a : (l, k) := (l − 1, T[l]);
else
    l0^b : (k, T[r], r) := (T[r], T[1], r − 1);
end if
l1 : while r ≥ 2 do
  [ l2 : (i, j) := (l, 2l);
    l3 : while j ≤ r do
        [ l4 : if j ≤ r − 1 ∧ T[j] < T[j + 1] then
              l4^a : j := j + 1;
          end if
          l5 : if k ≥ T[j] then
              l5^a : break;
          end if;
          l6 : (T[i], i, j) := (T[j], j, 2j); ]
    end while
    l7 : T[i] := k;
    l8 : if l ≥ 2 then
        l8^a : (l, k) := (l − 1, T[l]);
      else
        l8^b : (k, T[r], r) := (T[r], T[1], r − 1);
    end if
    l9 : T[1] := k; ]
end while
```

Fig. 3. HEAPSORT

$$
\begin{array}{rrr}
l - 1 \geq 0 & r \geq 2 & 2l - r \geq 0 \\
n \geq 3 & r \leq n & j = 2i
\end{array}
$$

Repeating the analysis assuming $l \geq 1$, we obtained the additional invariants $2l \leq j,\ 2l + 2r \geq j + 2$, matching the invariants generated using abstract interpretation [7].

Vagrant Robot

We analyzed a hybrid system modeling the position of a robot over time, taken from [12], represented as the linear transition system presented in Figure 4. The robot works in two alternating phases modeled by locations l_0 and l_1, each taking between 1 and 2 seconds. In l_0 it moves approximately northeast (both x and y increase), and in l_1 it moves approximately southeast (x increases and y decreases). The result after quantifier elimination is too large to be factorized or to be converted into some normal form by REDLOG. Therefore, we analyzed

$$V = \{x, y, t\}, L = \{l_0, l_1\}, T = \{\tau_1, \tau_2\}$$
$$\tau_1 : \langle l_0, l_1, \rho_1 \rangle \,,\ \tau_2 : \langle l_1, l_0, \rho_2 \rangle$$
$$\Theta : \ x = 0 \ \wedge \ y = 0 \ \wedge \ t = 0$$

$$\rho_1 : \begin{bmatrix} t' - t \le x' - x \le 2(t' - t) \ \wedge \\ t' - t \le y' - y \le 2(t' - t) \ \wedge \\ 1 \le t' - t \le 2 \end{bmatrix} \qquad \rho_2 : \begin{bmatrix} t' - t \le x' - x \le 2(t' - t) \ \wedge \\ -(t' - t) \ge y' - y \ge -2(t' - t) \ \wedge \\ 1 \le t' - t \le 2 \end{bmatrix}$$

Fig. 4. Transition system for the vagrant robot

the expression with each of the coefficients set to 0, thus looking for invariants involving at most two variables at a time. The invariants obtained for cutpoint l_0 were: $t \le x \le 2t$, and $-t \le y \le t$. Taking their post image produces the following invariants at location l_1: $t \le x \le 2t$, $-t + 2 \le y \le t + 2$, $y \le 2t$ and $t \ge 1$.

Note that the invariant assertions above are true immediately after a discrete mode transition is taken. If the continuous evolution is to be taken into account, as is always the case for a hybrid system, we need to perform an additional post image computation at each location to obtain the true invariant from a hybrid system point of view. In either case, the invariants above suffice to establish the unreachability of the point $y = 12, \ x = 9$, which is posed as a challenge to the reader in [12].

6 Conclusions

We have presented a method that generates invariants by solving for their coefficients directly. This is in contrast to traditional methods, which work iteratively toward a fixed point. The method generates constraints using *Farkas' Lemma*. Theoretically, the method is sound and complete, guaranteeing that any linear invariant of a linear program which is provable using an inductive linear assertion can be be found using our approach.

The main drawback of the method is that it produces non-linear constraints. As a result, while the technique can be applied to generate subtle invariants for small systems, its applicability to larger systems is limited. We are confident that, as the state of art in non-linear constraint solving advances, our method will become more and more practical.

References

1. F. Besson, T. Jensen, and J.-P. Talpin. Polyhedral analysis of synchronous languages. In *Static Analysis Symposium, SAS'99*, Lecture Notes in Computer Science 1694, pages 51–69, 1999.
2. A. Bockmayr and V. Weispfenning. Solving numerical constraints. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 12, pages 751–842. Elsevier Science, 2001.

3. E.M. Clarke. Synthesis of resource invariants for concurrent programs. In *ACM Principles of Programming Languages*, pages 211–221, January 1979.
4. G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, sep 1991.
5. G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In H.Brakhage, editor, *Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183, 1975.
6. P. Cousot and R. Cousot. Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Principles of Programming Languages*, pages 238–252, 1977.
7. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among the variables of a program. In *ACM Principles of Programming Languages*, pages 84–97, January 1978.
8. A. Dolzmann and T. Sturm. REDLOG: Computer algebra meets computer logic. *ACM SIGSAM Bulletin*, 31(2):2–9, June 1997.
9. K. Fukuda and A. Prodon. Double description method revisited. In *Combinatorics and Computer Science*, volume 1120 of *LNCS*, pages 91–111. Springer-Verlag, 1996.
10. N. Halbwachs and Y.-E. Proy. *POLyhedra desK cAlculator (POLKA)*. VERIMAG, Montbonnot, France, September 1995.
11. N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.
12. T.A. Henzinger and P.-H. Ho. Model-checking strategies for hybrid systems. In *Conference on Industrial and Engineering Applications of AI and Expert Systems*, 1994.
13. T.A. Henzinger and P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. In P. Wolper, editor, *Computer-Aided Verification*, LNCS 939, pages 225–238. 1995.
14. H. Hong. RISC-CLP(Real): Constraint logic programming over real numbers. In *CLP: Selected Research*. MIT Press, 1993.
15. J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Principles of Programming Languages(POPL)*, pages 111–119, January 1987.
16. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.
17. A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.
18. A. Tarski. A decision method for elementary algebra and geometry. *Univ. of California Press, Berkeley*, 5, 1951.
19. V. Weispfenning. Quantifier elimination for real algebra—the quadratic case and beyond. In *Applied Algebra and Error-Correcting Codes (AAECC) 8*, pages 85–101, 1997.
20. N. Wirth. *Algorithms + Data Structures = Programs*. Prentice-Hall, 1976.
21. R. Zippel. *Effective Polynomial Computation*. Kluwer Academic Publishers, Boston, 1993.