Non-linear Loop Invariant Generation using Gröbner Bases

Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna *
Department of Computer Science,
Stanford University,
Stanford, CA 94305,
USA.
{srirams,sipma,zm}@theory.stanford.edu

Abstract

We present a new technique for the generation of non-linear (algebraic) invariants of a program. Our technique uses the theory of ideals over polynomial rings to reduce the non-linear invariant generation problem to a numerical constraint solving problem. So far, the literature on invariant generation has been focussed on the construction of linear invariants for linear programs. Consequently, there has been little progress toward non-linear invariant generation. In this paper, we demonstrate a technique that encodes the conditions for a given template assertion being an invariant into a set of constraints, such that all the solutions to these constraints correspond to non-linear (algebraic) loop invariants of the program. We discuss some trade-offs between the completeness of the technique and the tractability of the constraint-solving problem generated. The application of the technique is demonstrated on a few examples.

Categories and Subject Descriptors: F.3.1 [Verifying Programs]: Invariants, I.1.2 [Algorithms]:Algebraic Algorithms, F.4.1 [Mathematical Logic]:Logic and Constraint Programming, F.3.2 [Semantics of Programming]: Program Analysis.

General Terms: Algorithms, Languages, Theory, Verification.

Keywords: Program Analysis, Verification, Invariant Generation, Symbolic Computation, Ideals, Gröbner Bases, Constraint Programming.

* 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, and by NAVY/ONR contract N00014-03-1-0939.

POPL'04, January 14–16, 2004, Venice, Italy.

1 Introduction

An *invariant* of a program at a location is an assertion that is true of any program state reaching the location. The importance of the automatic invariant generation problem for the verification, and the analysis of programs is well-known. Invariant assertions can be used directly to establish properties of a program, or used indirectly to obtain lemmas for proving other safety and liveness properties of the program [20].

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. Inductive assertions have been traditionally used to prove programs correct, starting with the Floyd-Hoare method of inductive assertions for verifying partial-correctness [14, 16]. It can be shown that any inductive assertion is invariant, and furthermore, all known techniques to establish invariants use inductive assertions. In this sense, invariant generation methods are, in fact, inductive assertion generation methods.

Traditionally, invariant generation has been performed using the iterative technique under the framework of abstract-interpretation [10, 11]. Since an invariant is true of every reachable state of a program, any over-approximation of the set of reachable states is an invariant. To generate such an over-approximation, the iterative technique starts from the initial states of the program, and iterates until no more states can be added. In order to converge, the technique uses a heuristic guess called *widening* to informally guess the limit of a sequence of iterative steps. However, widening may sometimes produce over-approximations that are too large to be useful. This technique has been applied with numerous refinements to generate invariants, especially linear invariants [18, 11, 15, 6, 26, 5, 4]. However, apart from a few exceptions [3, 22], there has been little progress on the generation of non-linear invariants involving multiplication. One of the reasons for this may be the lack of algorithmic improvements needed to define reasonable assertion manipulation techniques for the domain of non-linear assertions. Polyhedra, for instance, have been widely studied with advances in polyhedral invariant generation going hand in hand with algorithmic improvements in polyhedral manipulation techniques.

As an alternative to the iterative strategy, we have proposed a constraint-based technique to generate linear invariants [9]. The main idea there is to fix a template candidate assertion, and generate constraints on the coefficients in order to guarantee its inductiveness. Any solution to these constraints corresponds to an inductive assertion. The technique does not rely on widening heuristics, and can potentially generate stronger invariants than the iterative technique. However, so far, its use in practice has been limited to linear invariants of linear programs.

In this paper, we present a technique to generate algebraic inductive assertions of the form $p(x_1,\ldots,x_n) = 0$, where p is a polynomial of fixed degree with real coefficients. Similar to our previous work, we translate the invariant generation into a constraint solving problem using a theory that describes the consequences of assertions. However, the theory used for linear invariants is different from that needed for algebraic invariants. In the linear case, *Farkas' Lemma* [23] provides a straightforward way to encode the conditions for being an invariant. For the non-linear case, we demonstrate that Gröbner bases [12] can be used to reduce the invariant generation problem to a non-linear constraint solving problem that is shown to be in the parametric linear form [2]. We then discuss solution techniques to these constraints, and show that any solution to these constraints is an invariant, thus proving the soundness of our technique. Even though completeness seems achievable in theory, we find that insistence on completeness makes the constraint solving intractable. Hence, we present tractable relaxations that make our technique feasible. We demonstrate our technique on a few examples.

Gröbner bases are also used in the work of Müller-Olm and Seidl, which presents a backward propagation-based method for non-linear programs without branch conditions [22]. The use of widening/narrowing is avoided by observing that ideals in certain polynomial rings (called *Noetherian Rings*) satisfy the ascending chain condition. The technique is shown to be exact for polynomial constants, i.e, invariants of the form $x = p$, where x is a program variable and p is a polynomial in the program variables. However, their technique is applicable only to programs where all conditional branches are abstracted by non-deterministic choices. In contrast, our technique sacrifices completeness for the ability to handle algebraic branch conditions.

The rest of the paper is organized as follows: Section 2 presents some preliminary definitions along with a statement of the problem. The constraint generation technique is described in section 3. Techniques for solving these constraints are discussed in section 4. In section 5, the technique is illustrated with a few examples. Section 6 concludes with some ideas for future work. The appendix presents a few results alluded to in the paper.

2 Preliminaries

In this section, we introduce the theory of ideals, which is at the heart of our technique, and present our computational model of transition systems.

Algebraic Assertions

We begin by presenting some definitions and results from ideal theory. Informally, an ideal is a set of equations between polynomials along with all their consequences. Ideals can be used to deduce facts about the set of points that satisfy their underlying equations. Some of the results in this section are stated in general terms without proofs. Details can be found in most standard texts on algebra and algebraic geometry [12, 21].

Let $\mathfrak{R}$ be the set of reals and $\mathcal{C}$ be the set of complex numbers obtained as the *algebraic closure* of the reals. Let $\{x_1,\ldots,x_n\}$ be a set of variables. The set of polynomials on the variables, whose coefficients are drawn from the real (complex) numbers is denoted by $\mathfrak{R}[x_1,\ldots,x_n]$ ($\mathcal{C}[x_1,\ldots,x_n]$). We shall work with these rings even though the results can be applied to many other rings, including $Z[x_1,\ldots,x_n]$, the ring of integer polynomials.

Definition 1 (Algebraic Assertions) *An* algebraic assertion ψ *is an assertion of the form* $\bigwedge_i p_i(x_1,\ldots,x_n) = 0$ *where each* $p_i \in \mathfrak{R}[x_1,\ldots,x_n]$. *The* degree *of an assertion is the maximum among the degrees of the polynomials that make up the assertion.*

The set of points in the complex plane that satisfy an algebraic assertion is called a *variety*. Given an assertion $\psi \equiv \bigwedge_i (p_i(\vec{x}) = 0)$, its corresponding variety is defined as

$$\textbf{Variety}(\psi) = \{\vec{z} \in \mathcal{C}^n \mid (\forall i)\ p_i(\vec{z}) = 0\}$$

Definition 2 (Ideals) *A set $I \subseteq \mathfrak{R}[x_1,\ldots,x_n]$ is an ideal, if and only if*

1. *$0 \in I$,*
2. *If $p_1, p_2 \in I$ then $p_1 + p_2 \in I$,*
3. *If $p_1 \in I$ and $p_2 \in \mathfrak{R}[x_1,\ldots,x_n]$ then $p_1 p_2 \in I$.*

An ideal *generated by* a set of polynomials P, denoted by $((P))$ is the smallest ideal containing P. Equivalently,

$$((P)) = \left\{ g_1 p_1 + \cdots + g_m p_m \;\middle|\; \begin{array}{c} g_1,\ldots,g_m \in \mathfrak{R}[x_1,\cdots,x_n], \\ p_1,\ldots,p_m \in P \end{array} \right\}$$

An ideal I is said to be *finitely generated* if there is a finite set P such that $I = ((P))$. A famous theorem due to Hilbert states that all ideals in $\mathfrak{R}[x_1,\ldots,x_n]$ are finitely generated.

As a result, algebraic assertions can be seen as the generators of an ideal and vice-versa. Any ideal defines a variety, which is the set of the common zeros of all the polynomials it contains. This correspondence between ideals and varieties is one of the fundamental observations involving algebraic assertions called the *Hilbert's Nullstellensatz*. We state the relevant (and the easy) direction of this theorem below:

Theorem 1 (Hilbert's Nullstellensatz) *Consider an algebraic assertion ψ. Let $I = ((\psi))$ be the ideal generated by ψ in $\mathfrak{R}[x_1,\ldots,x_n]$, and $f(x_1,\ldots,x_n) \in \mathfrak{R}[x_1,\ldots,x_n]$. If $f \in I$, then*

$$(\forall \vec{z} \in \mathcal{C}^n)\ \psi(\vec{z}) \Rightarrow (f(\vec{z}) = 0)$$

i.e, $\psi(z) \models (f(z) = 0)$.

PROOF. Let $\psi \equiv f_1 = 0 \wedge \cdots \wedge f_m = 0$ and $f \in I$. Hence, we can express f as

$$f = g_1 f_1 + \cdots + g_m f_m$$

for some $g_1,\ldots,g_m \in \mathfrak{R}[x_1,\ldots,x_n]$. Let $\vec{z} \in \mathcal{C}^n$ be such that $\psi(\vec{z})$ holds. Then, $f(\vec{z}) = \sum_{i=1}^m (g_i(\vec{z}) f_i(\vec{z})) = 0$, since each $f_i(\vec{z}) = 0$. Therefore, we have shown that

$$(\forall \vec{z} \in \mathcal{C}^n)\ \psi(\vec{z}) \Rightarrow (f(\vec{z}) = 0)$$

□

The theorem shows that membership in an ideal I leads to semantic entailment in the variety induced by I. Hence, an ideal is (informally) a *Consequence-Closed* set of polynomials. While a similar statement can be made in the converse, it is not relevant to the overall soundness of our technique, and therefore, we omit it from the discussion. Note that even though the ideals are defined in $\Re[x_1,\ldots,x_n]$, the varieties along with the consequence relations are defined over the complex numbers, which subsume the reals. This is a technicality that arises from the fact that the reals are not algebraically closed.

Example 1 *Consider the ideal $I = ((\{x^2+1\}))$. The variety corresponding to this ideal is the set $V = \{i,-i\}$, which are the complex roots of unity. Any other member of the ideal can be expressed as $p\cdot(x^2+1)$, where p is an arbitrary polynomial over x. It can be easily seen that each of these polynomials is zero at the points where $x^2+1=0$, i.e,*

$$(\forall p \in I)\ p(i) = 0 \ \wedge\ p(-i) = 0$$

In the remainder of this subsection, we use the concept of ideals to derive a systematic algorithm for determining, for a given assertion ψ and a polynomial f, whether $f \in ((\psi))$. Assume that all the polynomials are drawn from $\Re[x_1,\ldots,x_n]$, and all consequence relations of the form $\psi_1 \models \psi_2$ are over the domain of complex numbers.

Let $X = \{x_1,\ldots,x_n\}$ be a set of variables. A *power-product* over X is of the form $x_1^{r_1}x_2^{r_2}\cdots x_n^{r_n}$, where each $r_i \in N$. The set of power products will be denoted by PP. A *term* (also a *monomial*) is of the form $c\cdot p$ where $c \in \Re$ and $p \in PP$. The set of terms will be denoted by *Term*.

Definition 3 (Term Orderings) *A* term ordering $<$ *is a total and strict ordering on PP that satisfies the following properties:*

1. $(\forall t \in PP)\ 1 \le t$,
2. *if* $(t_1 \le t_2)$ *then* $(\forall t \in PP) t_1 t \le t_2 t$.

These orderings can be extended to terms by ignoring the coefficients and just comparing the power-products.

Term orderings will be used to induce a reduction relation over polynomials. Many term orderings are used in the literature, the most common being the *lexicographic* and *total-degree lexicographic* orderings. We assume a linear ordering $\prec$ on the variables in X. Assuming that $x_1 \prec x_2 \prec \cdots \prec x_n$, the lexicographic extension $\prec_{lex}$ is defined as follows:

$$x_1^{r_1}\cdots x_n^{r_n} \prec_{lex} x_1^{q_1}\cdots x_n^{q_n} \text{ iff } (\exists i)\ r_i < q_i \ \wedge\ (\forall\ j < i)\ r_j = q_j$$

The ordering is lexicographic on the tuple $\langle r_1,\ldots,r_n\rangle$ corresponding to a term $x_1^{r_1}\cdots x_n^{r_n}$. A variant of this ordering is called the *total-degree* lexicographic ordering, which first compares power products by their *total degree*, defined as the sum of the powers of all the variables. For terms with the same total degree, the lexicographic ordering is used to resolve the tie. In general, the choice of ordering does have a bearing on the complexity of the algorithms, but such an effect is beyond the scope of this paper.

Given a polynomial g, we define its *lead term* (denoted $\mathrm{LT}(g)$) to be the largest among all its terms w.r.t. a given term-ordering.

Definition 4 (Reduction) *Let f, g be polynomials, and $<$ be a term-ordering. The reduction relation over polynomials, $\xrightarrow{g}$ is defined as: $f \xrightarrow{g} f'$ iff there exists term t in f s.t. $\mathrm{LT}(g)$ divides p, and*

$$f' = f - \frac{t}{\mathrm{LT}(g)} g$$

The reduction, in effect cancels out the term t that was selected. If no such reduction can be made, then f is said to be a normal-form w.r.t. $\xrightarrow{g}$. The reduction can also be extended to a finite set P of polynomials as $f \xrightarrow{P} f'$ iff $(\exists g \in P)\ f \xrightarrow{g} f'$. The reduction $\xrightarrow{P}$ can be shown to be terminating for any finite set of polynomials P as a direct consequence of the definition of term-orderings.

By $\overset{P}{\twoheadrightarrow}$, we denote the reflexive transitive closure of the relation $\xrightarrow{P}$. A normal form f of the reduction is a polynomial such that there is no further reduction that can be carried out on f. The reduction is said to be *confluent* if every polynomial reduces to a unique normal form. These definitions along with their properties are explained in standard textbooks on term-rewriting systems [1].

If $I = ((P))$, then the reduction relation induced by P can be used to check membership of a given polynomial f in I according to the following theorem:

Theorem 2 (Ideal Membership) *Let $I = ((P))$ be an ideal, and f be a polynomial. If $f \overset{P}{\twoheadrightarrow} 0$ then $f \in I$.*

PROOF. Proof proceeds by induction on the length of the derivation. It is trivially true for zero length derivations, since $0 \in I$. Let $f \xrightarrow{P} f' \overset{P}{\twoheadrightarrow} 0$. It follows that $f' = f - tg$, for some suitable term t, and some $g \in P$. Since $f' \in I$ (the induction hypothesis), and $g \in P$. Therefore $f' + tg \in I$. Thus, $f \in I$. □

Example 2 *Assume a set of variables x,y,z with a precedence ordering $x > y > z$. Consider the ideal $I = ((f : x^2-y,\ g : y-z,\ h : x+z))$, and the polynomial $p = x^2 - y^2$. We shall use the total lexicographic ordering. From the definition of the ordering relation, it follows that $x^2 > z$. The lead term in the polynomial $f :\ x^2 - z$ is x^2, which divides the term $t :\ x^2$ in p. Thus, $p \xrightarrow{f} p'$, where*

$$p' \;=\; \underbrace{(x^2-y^2)}_{p} \;-\; \underbrace{\frac{x^2}{x^2}}_{\frac{t}{\mathrm{LT}(f)}} \;\underbrace{(x^2-y)}_{f} \;= (-y^2+y)$$

The following sequence of reductions shows the membership of p in the ideal

$$p \xrightarrow{h} -zx - y^2 \xrightarrow{h} z^2 - y^2 \xrightarrow{g} -yz + z^2 \xrightarrow{g} -z^2 + z^2 \equiv 0$$

thus $p \overset{I}{\twoheadrightarrow} 0$, and hence $p \in I$. However, the reduction sequence

$$p \xrightarrow{f} -y^2 + y \xrightarrow{g} -yz + y \xrightarrow{g} -z^2 + y \xrightarrow{g} -z^2 + z$$

reaches a normal-form without showing the ideal membership.

Since the reduction $\xrightarrow{P}$ may not be confluent, it is not possible to use the theorem to decide membership using an arbitrary ideal basis P. However, given an ideal I, there is a special set of generators G such that $I = ((G))$, and the reduction relation $\xrightarrow{G}$ induced by G is confluent. Such a basis for I is variously called the *Gröbner Basis* or *Standard Basis* of I.

Theorem 3 (Gröbner Basis) *Let $I = ((P))$ be an ideal and f be a polynomial. Let G be the Gröbner basis of I. $f \overset{G}{\twoheadrightarrow} 0$ iff $f \in I$.*

PROOF. A proof of this theorem can be found in any standard text or survey on this topic [12, 21]. □

Since any reduction is terminating, Theorem 3 provides a decision procedure for ideal membership. We shall use $\text{NF}_G(p)$ to denote the normal form of a polynomial p under $\xrightarrow{G}\!\!\!\!\rightarrow$. The subscript G in $\text{NF}_G(p)$ may be dropped if it is evident from the context. The standard algorithm for computing the Gröbner basis of an ideal is known as the *Buchberger algorithm*. There are numerous implementations of this algorithm available with standard computer-algebra packages and polynomial computation libraries. As an example, the library GROEBNER implements many improvements over the standard algorithm [30].

Example 3 *Consider again the ideal from Example 2; $I = ((f : x^2 - y,\ g : y - z,\ h : x + z))$. The Gröbner basis for I is $G = \{f' : z^2 - z,\ g : y - z,\ h : x + z\}$. With this basis, every reduction of $p : x^2 - y^2$ will yield a normal form 0.*

Templates

Our technique for invariant generation aims to find polynomials which satisfy certain properties. To represent these sets of polynomials we use templates, which are polynomials with coefficients that are linear expressions over some set of template variables. In this subsection, we show that the theory of ideals can be naturally extended to templates. In particular, we show that there exist confluent reductions on templates that allow the generation of constraints on the template variables, such that the resulting set of polynomials is precisely the set of polynomials that belong to the corresponding ideal.

Definition 5 (Templates) *Let A be a set of* template variables *and $\mathcal{L}(A)$ be the domain of all* linear expressions *over variables in A of the form $c_0 + c_1a_1 + \ldots + c_na_n$, where each c_i is a real valued coefficient. A* template *over A, X is a polynomial over variables in X with coefficients drawn from $\mathcal{L}(A)$.*

Example 4 *Let $A = \{a_1, a_2, a_3\}$, hence $\mathcal{L}(A)$ is the set of expressions*

$$\mathcal{L}(A) = \{c_0 + c_1a_1 + c_2a_2 + c_3a_3 \mid c_0, \ldots, c_3 \in \mathfrak{R}\}$$

The set of templates lies in the ring $\mathcal{L}(A)[x_1, \ldots, x_n]$. As an example, consider the template

$$(2a_2 + 3)x_1x_2^2 + (3a_3)x_2 + (4a_3 + a_1 + 10)$$

Definition 6 (Semantics of Templates) *Given a set of template variables A, an A-environment (if A is clear from the context, then simply an environment) is a map α that assigns real values to each variable in A. Hence, this map can be naturally extended to map expressions in $\mathcal{L}(A)$ to their corresponding values in $\mathfrak{R}$, and to map polynomials in $\mathcal{L}(A)[x_1, \ldots, x_n]$ to their corresponding polynomials in $\mathfrak{R}[x_1, \ldots, x_n]$.*

Example 5 *The environment $\alpha \equiv \langle a_1 = 0, a_2 = 1, a_3 = 2\rangle$, maps the template*

$$(2a_2 + 3)x_1x_2^2 + (3a_3)x_2 + (4a_3 + a_1 + 10)$$

from Example 4 to the polynomial

$$5x_1x_2^2 + 6x_2 + 18$$

The reduction $\xrightarrow{g}$ for polynomials can be extended to a reduction for templates in a natural way.

Definition 7 (Reduction of Templates) *Let p be a polynomial in $\mathfrak{R}[x_1, \ldots, x_n]$ and f, f' be templates over A and $\{x_1, \ldots, x_n\}$. The reduction relation is defined as: $f \xrightarrow{p} f'$ iff the lead term $\text{LT}(p)$ divides a term $c \cdot t$ in f with coefficient $c(a_0, \ldots, a_m)$ and*

$$f' = f - \frac{c \cdot t}{\text{LT}(p)} p$$

Note that the reduction is defined to be the same as the reduction relation over polynomials. This can, in turn, be extended to sets of polynomials to define reductions $\xrightarrow{G}$ over templates for sets of polynomials G. Henceforth, we shall use the symbols f, g with subscripts to denote templates and the symbols h, p to denote polynomials.

Example 6 *Let p be the polynomial $x^2 - y$, with $\text{LT}(p) = x^2$. Consider the template*

$$f :\ ax^2 + by^2 + cz^2 + dz + e$$

The lead term $\text{LT}(p)$ divides the term ax^2 in f. Therefore, $f \xrightarrow{p} f'$, where

$$f' : (ax^2 + by^2 + cz^2 + dz + e) - \frac{ax^2}{x^2}(x^2 - y) = by^2 + cz^2 + dz + e + ay$$

Given a template f and an ideal I, we are interested in finding those environments α such that $\alpha(f) \in I$. We achieve this by obtaining constraints on the environment variables A such that any solution α satisfies $\alpha(f) \in I$.

The properties of the Gröbner basis reduction over polynomials can be extended smoothly to templates. Proofs of these results can be found in the appendix. We first show that the extension of reduction is consistent w.r.t. the semantics of templates under any A-environment. The confluence of Gröbner basis reduction over templates, guarantees that a unique normal form exists for any template (Theorem 11). The template membership theorem (Theorem 13) proves that if $f_1 = \text{NF}_G(f)$ is the normal form of f, then for each environment α, $\alpha(f) \in ((G))$ iff $\alpha(f_1)$ is identically zero.

Theorem 4 (Zero Polynomial Theorem) *A polynomial p is zero for all the possible values of $x_1, \ldots, x_n$ iff all its coefficients are identically zero.*

PROOF. Proof proceeds by induction on the number of variables n. For $n = 1$, the result is immediate from the fundamental theorem of algebra, restricting the number of roots of a non-zero univariate polynomial of degree n. Therefore, a non-zero polynomial cannot be zero everywhere. Let us assume that the result is true for all polynomials in $n-1$ variables. Let $p = p_0 + p_1x_n + \ldots + p_mx_n^m$, where each p_i is a polynomial in $x_1, \ldots, x_{n-1}$. For any value $\alpha_1, \ldots, \alpha_{n-1}$ of these $n-1$ variables, the polynomial $p = p_0(\alpha_1, \ldots, \alpha_{n-1}) + p_1(\alpha_1, \ldots, \alpha_{n-1})x_n + \cdots + p_m(\alpha_1, \ldots, \alpha_{n-1})x_n^m$ is identically zero for all x_n and hence each p_i is identically zero for all possible values $\alpha_1, \ldots, \alpha_{n-1}$. Using the inductive hypothesis, every coefficient in each p_i is zero, and hence, all the coefficients of p are zero. □

Given a template f and an ideal I with Gröbner basis G, we first compute $\text{NF}_G(f)$, and then, equate each coefficient of the normal form to zero to obtain a set of equations over the template variables A. Any solution to this set of equations yields an A environment α such that $\alpha(f) \in I$ (and conversely).

Example 7 *Let I be the ideal $((x^2 - y,\ y - z,\ z + x))$ of Example 2 with Gröbner basis $G = \{x + z,\ y - z, z^2 - z\}$. We are interested in polynomials represented by the template*

$$ax^2 + by^2 + cz^2 + dz + e$$

that are members of the ideal. The normal form of this template is

$$(a + b + c + d)z + e$$

Equating each coefficient in the normal form to zero, we obtain

$$\begin{array}{l} a + b + c + d = 0 \\ e = 0 \end{array}$$

From this, we can generate all instances of the template that belong to I. Some examples are $x^2 - y^2$, $y^2 - z$, $2x^2 - z^2 - z$. On the other hand, $x^2 + y^2 - z \notin I$, as the coefficients do not satisfy the constraints.

Transition Systems and Invariants

We begin by defining transition systems in general.

Definition 8 (Transition System) *A* transition system *is a tuple $\langle V, L, \mathcal{T}, \ell_0, \Theta\rangle$, where V is a set of* variables*, L is a set of* locations*, $\mathcal{T}$ is a set of* transitions*. A state s is an interpretation of the variables in V. Each transition $\tau \in \mathcal{T}$ is a tuple $\langle l_1, l_2, \rho_\tau\rangle$, where l_1 and l_2 are the* pre- *and* post- *locations of the transition. The* transition relation *ρ_τ is a first-order assertion over $V \cup V'$, where V denotes the* current-state *variables and V' denotes the* next-state *variables. The location $\ell_0 \in L$ is the* initial location*, and Θ is a first-order assertion over V denoting the* initial condition.

Transition systems are the standard representation for many types of programs. A more detailed presentation of these systems can be found in [20]. Given a transition system, we define its transition graph with locations labeled by vertices and edges labeled by transitions, such that each edge connects the pre-location and the post-location of its labeling transition. Given a path π in this graph, we define the transition relation ρ_π corresponding to π as the relational composition of all the transition relations labeling the edges along the path.

Definition 9 (Invariant) *Let $P \equiv \langle V, L, \mathcal{T}, l_0, \Theta\rangle$ be a transition system. An* invariant *at a location $l \in L$ is defined as an assertion ψ over V, such that ψ holds on all the states that can be reached at location l. An invariant of the transition system is an assertion ψ which holds at all the locations of the transition system.*

Given a *domain of assertions $\mathcal{D}$*, an *assertion-map* for a transition system is a map $\eta : L \mapsto \mathcal{D}$ that associates each location of the transition system with an assertion.

Definition 10 (Inductive Assertion Map) *An assertion map η is said to be inductive iff the following conditions hold:*

Initiation *The assertion at l_0 subsumes the initial condition,*

$$\Theta \models \eta(l_0)$$

Consecution *For each transition τ from l_i to l_j,*

$$\eta(l_i) \wedge \rho_\tau \models \eta(l_j)'$$

$$\begin{array}{l} \textbf{integer } i, j, s \textbf{ where}(s = 0 \wedge j = j_0) \\ l_0 :\ \textbf{while } (\cdots) \textbf{ do} \\ \quad l_1 :\ (s, j) := (s + i, j - 1) \end{array}$$

Figure 1. A program to multiply two numbers.

It is a well known fact from the pioneering work of Floyd and Hoare [14, 16], that if η is an inductive assertion map then $\eta(l)$ is invariant at l. Furthermore, for general transition systems, given any invariant φ at l, there is an inductive assertion map η such that $\eta(l)$ implies the invariant φ. In fact, all known invariant generation methods are inductive assertion generation methods.

Algebraic Transition Systems

We now specialize general transition systems into algebraic transition systems.

Definition 11 (Algebraic Transition System) *An* algebraic transition system *is a transition system $\langle V, L, \mathcal{T}, \ell_0, \Theta\rangle$, such that, for each transition τ, the transition relation ρ_τ is an algebraic assertion over $V \cup V'$, and the initial condition Θ is an algebraic assertion over V.*

Example 8 *Consider the loop program to multiply two numbers shown in Figure 1. The corresponding transition system is given by*

$$\begin{array}{l} V = \{i, j, s, j_0\} \\ L = \{l_0\} \\ \mathcal{T} = \{\tau_1\},\ \textit{where} \\ \tau_1 : \left\langle l_0, l_0, \left[\begin{array}{l} s' - s - i = 0 \\ j' - j + 1 = 0 \\ i' - i = 0 \\ j_0' - j_0 = 0 \end{array}\right] \right\rangle \\ \ell_0 = l_0 \end{array}$$

which is an algebraic transition system.

For algebraic transition systems in general, the composition of transition relations along a path may lie outside the domain of algebraic transitions. However, if the transition relations are *separable*, that is, each variable in V' is expressed as a polynomial expression over the variables in V, we can compose relations along any path. For instance, the transition system in Example 8 is separable.

Corresponding to general assertion maps, we define an *algebraic assertion map* η, wherein each location l is mapped to an assertion $\eta(l)$ of the form $p = 0$. We shall use $\eta(l)$ to denote both the assertion $p = 0$ and the polynomial p.

3 Constraint Generation

In this section, we present the basic algorithm for invariant generation in algebraic transition systems. Given an algebraic transition system, we first define a template map that maps each location to a template over a set of abstract coefficient variables A. We then generate constraints on all the template variables A, guaranteeing that any solution to these constraints corresponds to an inductive assertion map.

Definition 12 (Template Map) *Let $P \equiv \langle V, L, \mathcal{T}, l_0, \Theta\rangle$ be an algebraic transition system. Assuming a set of* template variables *A, an* invariant template *over P is a map* $\eta: L \mapsto \mathcal{L}(A)[V]$, *that maps each location in L to a template over A.*

When the variables in A are instantiated to real values, the invariant template is then *specialized* to a polynomial assertion map. The problem that we intend to tackle in this section is as follows: Given an algebraic transition system with a template map, compute constraints on the template variables A, such that any solution to these constraints specializes the template to a valid inductive assertion map.

The solution to the problem involves encoding the initiation and the consecution conditions that any inductive assertion map must satisfy. In practice, if the transitions can be composed, it is more effective to do this over partial maps by selecting a suitable set of cutpoints.

The example below shows a (rather lengthy) template that will be used as a running example.

Example. Consider the example transition system in Example 8. Since there is one location l_0, we set the $\eta(l_0)$ to be a *generic quadratic* form on $\{s, i, j, j_0\}$ as shown below:

$$\eta(l_0): \left[\begin{array}{c} a_0 s^2 + a_1 s + a_2 si + a_3 sj + a_4 s j_0 + a_5 i^2 + \\ a_6 i + a_7 ij + a_8 i j_0 + a_9 j^2 + \\ a_{10} j + a_{11} j j_0 + a_{12} j_0^2 + a_{13} j_0 + a_{14} \end{array}\right]$$

Encoding Initiation

Initiation is expressed by encoding the membership of $\eta(l_0)$ to the ideal generated by the initial assertion Θ, i.e., $\eta(l_0) \in ((\Theta))$ which in turn implies $\Theta \models \eta(l_0)$ by the Nullstellensatz. The following are the steps involved in the encoding:

1. Compute the Gröbner basis G for $((\Theta))$,
2. Reduce the template $\eta(l_0)$ using G, to compute $f = \mathrm{NF}(\eta(l_0))$,
3. For each term in f, equate the coefficient expression of the term to zero, in order to obtain a constraint. The overall initiation constraint is the conjunction of all the constraints thus obtained.

Example 9 *We now generate initiation constraints for the system in Example 8. The Gröbner basis of* $\Theta:\ s = 0 \ \wedge\ j - j_0 = 0$ *is* $G = \{s, j - j_0\}$. *The normal form of* $\eta(l_0)$ *w.r.t. G is*

$$f = a_5 i^2 + a_6 i + (a_7 + a_8) i j_0 + (a_9 + a_{11} + a_{12}) j_0^2 + (a_{10} + a_{13}) j_0 + a_{14}$$

Setting each coefficient expression to zero, we obtain the constraints

$$\begin{array}{l} a_5 = a_6 = a_{14} = 0 \\ a_7 + a_8 = 0 \\ a_9 + a_{11} + a_{12} = 0 \\ a_{10} + a_{13} = 0 \end{array}$$

Encoding Consecution

For each location l_i and for each transition $\tau: \langle l_i, l_j, \rho\rangle$, consecution can be expressed by the implication

$$(\eta(l_i) = 0) \ \wedge\ \rho \ \models\ (\eta(l_j)' = 0)$$

There are two cases to consider here, one when the antecedent $\eta(l_i) = 0 \ \wedge\ \rho$ is satisfiable, i.e., τ is an *enabled transition*; the other case occurs when the antecedent is unsatisfiable, i.e., τ is *disabled*.

The disabled case consists of encoding

$$\rho \models \eta(l_i) \neq 0$$

This is achieved by computing the normal form of the reduction of $\eta(l_i)$ by the Gröbner basis of $((\rho))$. The normal form is set to be identical to a scalar different from zero. Thus all the terms in this normal form except the constant term have their coefficients set to zero, and the constant term is required to be non-zero. The rest of the section deals with the enabled case.

An Exact but Impractical Encoding

The first approach to encoding the enabled case of the consecution condition is similar to that of the initiation condition, i.e., to compute the Gröbner basis of the ideal generated by $\eta(l_i) = 0 \wedge \rho_\tau$. The normal form of $\eta(l_j)'$ is computed and constraints on the template variables that ensure that this normal form is identical to zero are obtained. However, the assertion $\eta(l_i) = 0 \wedge \rho_\tau$ contains a template. Hence, a variant of the Gröbner basis construction, resulting in a basis known as the *Comprehensive* Gröbner basis [28], is required.

The comprehensive basis is a set of condition-basis pairs (ψ, G), where ψ is a non-linear constraint on the template variables, and G is a Gröbner basis involving template polynomials with non-linear coefficient expressions. For each such pair, computing the normal form of $\eta(l_j)'$ under G, and equating each coefficient of each normal form to zero yields constraints ψ_1 over the template variables. The overall constraint for the pair is $\psi \Rightarrow \psi_1$. Unfortunately, this approach, though exact, is impractical, because the number of condition-basis pairs produced on a generic polynomial template is large, and the nonlinear constraints produced make the constraint solving problem intractable, even for simple programs.

Hence, we resort to an alternative approach that uses a stronger consequence relation to avoid the explicit construction of the comprehensive Gröbner basis but in doing so, we sacrifice completeness.

A Practical Alternative

The original consecution condition for a the assertion map η requires that the values after the transition are zero whenever the values before the transition is taken are zero. We define three increasingly stronger relations on the values of the invariant before and after the transition.

Definition 13 (Alternative Consequence Relations) *Let τ be the transition $\langle l_i, l_j, \rho\rangle$ and η be an algebraic assertion map. We define the following increasingly stronger notions of consecution:*

1. η *satisfies* polynomial-scale *consecution (PS) for τ iff there exists a polynomial p such that*

$$\rho \ \models\ (\eta(l_j)' - p\eta(l_i) = 0)$$

2. η *satisfies* constant-scale *consecution (CS) for τ iff there exists a real valued* parameter λ *s.t.*

$$\rho \ \models\ (\eta(l_j)' - \lambda\eta(l_i) = 0)$$

3. η *satisfies* constant-value *consecution (CV) for* τ *iff*

$$\rho \models (\eta(l_j)' - \eta(l_i) = 0)$$

4. η *satisfies* local-consecution *(LC) for* τ *iff*

$$\rho \models (\eta(l_j)' = 0)$$

LC consecution states that $\eta(l_j)'$ is zero upon taking the transition, independent of $\eta(l_i)$. CV consecution encodes the fact that the numerical value of the assertion does not change through the transition. The CS consecution on the other hand encodes the fact that the numerical value of the assertion after the transition is taken is a constant multiple of the numerical value prior to the transition. Each consecution can be seen as a generalization of the next consecution. For instance, CV consecution can be seen as CS consecution for $\lambda = 1$. Similarly, LC consecution is again a special case of CS for $\lambda = 0$. CS consecution results when the polynomial p in PS consecution is of degree 0. We now show that any assertion map that satisfies PS consecution also satisfies (exact) consecution. This implies similar results for CV and CS consecutions.

Theorem 5 *Let* $\tau : \langle l_i, l_j, \rho \rangle$ *be a transition, and* η *be a polynomial assertion map.*

1. *If* η *satisfies CV consecution for* τ *then it satisfies CS consecution.*
2. *If* η *satisfies CS consecution for* τ *then it satisfies PS consecution.*
3. *If* η *satisfies PS consecution for* τ *then it satisfies (exact) consecution.*

PROOF. (1) and (2) follow directly from the definitions. We shall prove (3). Since η satisfies PS, $\exists\, p$ such that,

$$\rho \models (\eta(l_j)' - p\eta(l_i) = 0)$$

Assuming that $\eta(l_i) = 0 \wedge \rho$, we obtain

$$\eta(l_j)' - p \cdot \eta(l_i) = \eta(l_j)' - p \cdot 0 = \eta(l_j)' = 0$$

Thus, assuming $\eta(l_i) = 0 \wedge \rho$, we obtain $\eta(l_j)' = 0$. This shows that η satisfies consecution w.r.t. τ. $\square$

Thus, if an assertion map satisfies initiation, and any of the three restricted consecutions for each transition, then it is an inductive assertion map. In practice, we find that using CS consecution produces useful invariants without making the constraint solving problem intractable. The following theorem suggests a method to encode CS consecution:

Theorem 6 *Let* $G = \{g_1, \ldots, g_m\}$ *be the Gröbner basis of assertion* ψ*, and* p_1, p_2 *be polynomials. Let* $p_i' = \mathrm{NF}_G(p_i)$*, for* $i = 1,2$ *and* λ *be a real-valued parameter. If* $p_1' - \lambda p_2' = 0$*, then*

$$\psi \models (p_1 - \lambda p_2 = 0)$$

The enabled case of consecution is encoded using CS consecution as follows:

1. Let $f = \mathrm{NF}_G(\eta(l_i))$ and $g = \mathrm{NF}_G(\eta(l_j)')$, where G is the Gröbner basis of ρ.
2. Introducing a real parameter λ, we set each coefficient of the polynomial $\lambda f - g$ to zero, obtaining constraints involving the template variables, and the parameter λ.
3. The resulting constraint involving the template variables, and the parameter λ is existentially quantified by λ.

The overall constraint for the consecution of τ is a disjunction of the constraints for the enabled case and that for the disabled case. We illustrate this by encoding the consecution for our running example.

Example 10 *Returning to the transition system in Example 8, The only transition is* τ_1*, with transition relation*

$$\rho_{\tau_1} : \begin{bmatrix} s' - s - i = 0 \\ j' - j + 1 = 0 \\ i' - i = 0 \\ j_0' - j_0 = 0 \end{bmatrix}$$

which corresponds to the Gröbner basis $\{s' - s - i, j' - j + 1, i' - i, j_0' - j_0\}$*, generated under the total-degree lexicographic ordering with the precedence*

$$s' > j' > i' > j_0' > s > j > i > j_0$$

The normal form of $\eta'(l_0)$ *is given by*

$$f' = \begin{bmatrix} a_0 s^2 + (2a_0 + a_2)si + (a_1 - a_3)s + a_3 sj + \\ a_4 s j_0 + (a_0 + a_2 + a_5)i^2 + (a_1 - a_3 + a_6 - a_7)i + \\ (a_3 + a_7)ij + (a_4 + a_8)ij_0 + a_9 j^2 + \\ (a_{10} - 2a_9)j + a_{11} j j_0 + a_{12} j_0^2 + \\ (a_{13} - a_{11})j_0 + (a_9 - a_{10} + a_{14}) \end{bmatrix}$$

The template $\eta(l_0)$ *remains unaltered by the reduction. Hence, setting*

$$(\exists \lambda)\ \mathrm{NF}(\eta(l_0)') = \lambda \cdot \mathrm{NF}(\eta(l_0))$$

gives us the following constraints for the enabled case:

$$\begin{array}{l} a_0 = \lambda a_0 \\ 2a_0 + a_2 = \lambda a_2 \\ a_0 + a_2 + a_5 = \lambda a_5 \\ a_1 - a_3 = \lambda a_1 \\ a_1 - a_3 + a_6 - a_7 = \lambda a_6 \\ a_3 = \lambda a_3 \\ a_3 + a_7 = \lambda a_7 \\ a_4 = \lambda a_4 \\ a_4 + a_8 = \lambda a_8 \\ a_9 = \lambda a_9 \\ a_{10} - 2a_9 = \lambda a_{10} \\ a_9 - a_{10} + a_{14} = \lambda a_{14} \\ a_{11} = \lambda a_{11} \\ a_{13} - a_{11} = \lambda a_{13} \\ a_{12} = \lambda a_{12} \end{array}$$

The solution to these constraints is discussed in the next section. As mentioned above, the normal form of $\eta(l_0)$ *w.r.t. the Gröbner basis for* ρ *is itself, thus yielding the following constraints for the disabled case.*

$$a_1 = a_2 = \cdots = a_{13} = 0 \wedge a_{14} \neq 0$$

The overall constraint for a template map being inductive is given by the conjunction of the initiation and the consecution constraints. Let φ_Θ be the initiation constraint and φ_τ be the constraint corresponding to the consecution for transition τ. Then the overall constraints φ are given by

$$\varphi \equiv \varphi_\Theta \wedge \bigwedge_{\tau \in \mathcal{T}} \varphi_\tau$$

Let $[[\eta(l_i)]] = \alpha(\eta(l_i))$ denote the specialization of $\eta(l_i)$ by some solution α of φ, and let $[[\eta]] = \alpha \circ \eta$.

Theorem 7 (Soundness) *For any solution α of φ, the instantiation of the template map η with the solution values is an inductive assertion map.*

PROOF. Since the solution set satisfies φ, it must satisfy each individual φ_τ along with φ_Θ. Since the constraint set satisfies the φ_Θ, the normal form of $[[\eta(l_0)]]$ is identically zero. By Theorem 8, $[[\eta(l_i)]] \in ((\Theta))$, and by Theorem 1, $\Theta \models [[\eta(l_i)]] = 0$. Thus initiation holds.

Similarly, the constraints for consecution ensure that for some λ,

$$[[\mathrm{NF}(\eta(l_j)')]] - \lambda[[\mathrm{NF}(\eta(l_i))]] = 0$$

This implies

$$\rho_\tau \models [[\eta(l_j)']] - \lambda[[\eta(l_i)]] = 0$$

by Theorem 6. Therefore, $[[\eta]]$ satisfies CS consecution for each transition τ. Thus we have shown that $[[\eta]]$ satisfies the conditions of initiation and consecution for each transition. Therefore $[[\eta]]$ is an inductive assertion map. $\square$

The converse of this theorem (completeness) does not hold due to our choice of a stronger consecution condition that is satisfied by fewer inductive assertion maps. So far, we have reduced the invariant generation problem for algebraic transition systems to a set of constraints, such that any solution to these constraints forms an invariant. Solution methods to these constraints are discussed in the next section.

The complexity of the constraint generation process is linear in the size of the transition relation and the number of template variables. The Gröbner bases need to be computed only over initial conditions and transitions, which can be done efficiently. The number of Gröbner basis computations is linear in the program size. The reduction process can also be done efficiently, the number of reductions required being roughly linear in the size of the template to be reduced, if the template is a generic polynomial of a fixed degree.

4 Solving Constraints

In this section, we discuss techniques for solving the system of equations generated by the method described in Section 3. The constraints corresponding to the initiation and the disabled case for consecution are linear whereas the constraints corresponding to the enabled case of consecution vary depending on the consecution relation used. For the LC and CV cases, these constraints are linear equalities. For the CS consecution the constraints are non-linear, more specifically *Parametric Linear Constraints*. We shall first discuss the structure of these constraints in detail and then discuss solution techniques. We do not discuss techniques for the more general PS consecution. The constraint types are summarized in Figure 2.

Definition 14 (Parametric Linear Constraint) *Let A be the set of abstract variables and Λ be a set of multiplier variables. A parametric linear constraint is of the form*

$$l_0 + \lambda_1 l_1 + \cdots + \lambda_m l_m = 0$$

where, $l_0, \ldots, l_m \in \mathcal{L}(A)$ are linear expressions in A and $\Lambda = \{\lambda_1, \ldots, \lambda_m\}$, $\lambda_i \in \Re$.

A *linear constraint* is of the form $l_0 = 0$. Similarly, a *transform* is of the form

$$c_0 + c_1\lambda_1 + \cdots + c_m\lambda_m = 0$$

where, $c_0, \ldots, c_m \in \Re$. The constraint is *factorizable* iff $l_0 = c_1 l_1 = \cdots = c_m l_m$ for some real constants $c_1, \ldots c_m$. In such a case we express the constraint as

$$l_0 + \lambda_1 l_1 + \cdots + \lambda_m l_m = l_0(1 + c_1\lambda_1 + \cdots + c_m\lambda_m)$$

a product of a linear constraint and a transform constraint. The constraint generation algorithm provides us with a set of constraints. Some of the constraints (initiation constraints) are linear whereas the constraints from consecution are parametric linear constraints. The parameters in Λ are all existentially quantified. In the remainder of this section we discuss some techniques to eliminate the quantifiers.

Elimination by Splitting

The simplest technique for elimination is a CLP-style [17] elimination algorithm that maintains the linear constraints in a *constraint store* and repeatedly linearizes the remaining constraints. The main advantage of this technique is its efficiency. It has been observed to be fast and memory efficient in practice. Furthermore, it preserves the parametric linear form throughout the elimination process. On the other hand, it may not be able to remove all instances of the parameter. In such a case, we resort to more general elimination techniques. The following is a brief sketch of the technique:

Constraint-Store The constraint store is a set of linear constraints over the variables in A. We store these constraints in terms of a matrix that is always kept in a reduced row form using the standard Gaussian elimination. The operations supported by the store include the addition of a new constraint and the simplification of a given parametric linear constraint to a normal form.

The following are the major steps involved:

1. Each linear constraint is added to the constraint store and each transform constraint is used to eliminate one of the multipliers involved in the constraint system by rewriting it in terms of the remaining parameters.
2. Each factorizable constraint leads to a *split* into two cases, one where the linear constraint is added to the store, and the other where the transform constraint is applied to remove one multiplier from the system. Care must be taken to avoid inconsistent branches on a split. For instance a split on $a \neq 0$ in a branch should not be followed by a split on $a = 0$.

The steps shown above are repeated until the branch is unsatisfiable or all the constraints have been linearized. In almost all the observed cases, a majority of the branches are completely linearized, and only a few branches with unresolved parametric linear constraints remain. The latter can be resolved by generating more linear constraints as "hints" to help simplify the non-linear constraints further. These constraints can be obtained by choosing paths π starting from l_0 ending in l and encoding the condition

$$\Theta \wedge \rho_\pi \models \eta(l)'$$

where ρ_π represents the composition of the transition relations along π. Soundness can be shown to be preserved by these operations. Furthermore, the solution set can also be shown to be preserved.

Condition	Restriction	Constraint types
Initiation		linear equalities
Consecution	Local (LC)	linear equalities
	Constant Value (CV)	linear equalities
	Constant Scale (CS)	parametric linear
	Polynomial Scale (PS)	non-linear algebraic

Figure 2. Constraints obtained from different conditions for inductive assertions

Example 11 *To illustrate the technique, we solve the constraints for the enabled case of consecution for Example 8, recalled below from Example 10.*

$$\begin{array}{l} a_0 = \lambda a_0 \\ 2a_0 + a_2 = \lambda a_2 \\ a_0 + a_2 + a_5 = \lambda a_5 \\ a_1 - a_3 = \lambda a_1 \\ a_1 - a_3 + a_6 - a_7 = \lambda a_6 \\ a_3 = \lambda a_3 \\ a_3 + a_7 = \lambda a_7 \\ a_4 = \lambda a_4 \\ a_4 + a_8 = \lambda a_8 \\ a_9 = \lambda a_9 \\ a_{10} - 2a_9 = \lambda a_{10} \\ a_9 - a_{10} + a_{14} = \lambda a_{14} \\ a_{11} = \lambda a_{11} \\ a_{13} - a_{11} = \lambda a_{13} \\ a_{12} = \lambda a_{12} \end{array}$$

Factorizing, $(1-\lambda)a_0 = 0$*, we get* $1-\lambda = 0$ *or* $a_0 = 0, 1-\lambda \neq 0$*. In the first case, we rewrite all occurrences of* λ *by* 1*, resulting in the constraints*

$$\begin{array}{l} a_0 = a_2 = a_3 = a_4 = a_9 = a_{10} = a_{11} = 0 \\ a_1 - a_7 = 0 \end{array}$$

Repeating the strategy on the other branch, we obtain

$$\begin{array}{l} a_0 = a_3 = a_4 = a_9 = a_{11} = a_{13} = 0 \\ 2a_0 + a_2 = \lambda a_2 \\ a_0 + a_2 + a_5 = \lambda a_5 \\ a_1 - a_3 = \lambda a_1 \\ a_1 - a_3 + a_6 - a_7 = \lambda a_6 \\ a_3 + a_7 = \lambda a_7 \\ a_4 + a_8 = \lambda a_8 \\ a_{10} - 2a_9 = \lambda a_{10} \\ a_9 - a_{10} + a_{14} = \lambda a_{14} \\ a_{13} - a_{11} = \lambda a_{13} \end{array}$$

By simplifying and repeating the strategy, we finally obtain

$$a_0 = a_1 = \cdots = a_{14} = 0$$

Thus combining with the initiation constraints, we obtain

$$\begin{array}{l} a_{\{0,2,3,4,5,6,9,10,11,12,13,14\}} = 0 \\ a_1 - a_7 = 0 \\ a_7 + a_8 = 0 \end{array}$$

This in turn simplifies to yield: $a_1 = a_7 = -a_8$*, while all the other coefficients are zero. This corresponds to the invariant*

$$s = i(j_0 - j)$$

This invariant establishes the partial correctness of the program shown in example 8.

Generic Elimination Techniques

If the simple elimination by factorization technique fails, we resort to more general techniques for quantifier elimination. We summarize the viable approaches to this problem. The first generic elimination technique casts the constraints obtained as a matrix equation

$$B\vec{a} = 0$$

where $\vec{a}$ is a vector of variables from A, and B is a parametric matrix whose entries are linear expressions over the parameters. As a rule, the matrices involved are sparse in terms of the number of non-zero entries involving the parameters. The resulting problem can be treated using Gaussian Elimination with a few modifications. There has been some attention to solving these types of constraints [24, 2].

In contrast to constraints tailored to parametric linear constraints, more general quantifier elimination techniques may be used. Since the constraints are non-linear with real-valued variables, we can use real quantifier elimination tools to solve these constraints. A brief summary of the related work on this topic follows.

Tarski [25], established the decidability of quantifier elimination over the theory of reals with multiplication. However, the algorithm suggested by Tarski is non-elementary. This was remedied by Collins [7] using a technique called *Cylindric Algebraic Decomposition*. Collins and Hong [8] present an efficient version of this technique called *Partial Cylindric Algebraic Decomposition*, implemented in the tool QEPCAD. An alternative approach called the elimination at test point technique is taken by Weispfenning [27, 29]. The method is efficient over low degree polynomials and has been implemented in the tool REDLOG [13]. Even though these methods have high time and space complexities, we find that for most constraints we do not require these generic elimination techniques.

5 Applications

To show the viability of our approach, we present some application examples.

Generalized Readers-Writers

Consider the following transition system with variables r, w, k, k_0, c_1, c_2, modeling a generalization of the readers-writers problem.

$$\begin{array}{l} V = \{r, w, k, c_1, c_2, k_0\} \\ L = \{l_0\} \\ \mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4\} \\ \tau_1 : \left\langle l_0, l_0, \left(\begin{array}{l} w = 0 \\ r' = r + 1 \\ k' = k - c_1 \\ id(w, c_1, c_2, k_0) \end{array} \right) \right\rangle \end{array}$$

$$\tau_2: \left\langle l_0, l_0, \begin{pmatrix} r = 0 \\ w' = w+1 \\ k' = k - c_2 \\ id(r, c_1, c_2, k_0) \end{pmatrix} \right\rangle$$
$$\tau_3: \left\langle l_0, l_0, \begin{pmatrix} w = 0 \\ r' = r-1 \\ k' = k + c_1 \\ id(w, c_1, c_2, k_0) \end{pmatrix} \right\rangle$$
$$\tau_4: \left\langle l_0, l_0, \begin{pmatrix} r = 0 \\ w' = w-1 \\ k' = k + c_2 \\ id(r, c_1, c_2, k_0) \end{pmatrix} \right\rangle$$
$$\Theta: (r = 0 \wedge w = 0 \wedge k = k_0)$$
$$\ell_0 = l_0$$

The number of readers and writers are represented by r and w respectively. Initially we assume $k = k_0$ tokens to be present. Transition τ_1 models a reader obtaining access by checking if no writers are present, and obtaining c_1 tokens. Similarly transition τ_2 models a writer entering by obtaining c_2 tokens. Transition τ_3, τ_4 model the readers and the writers giving up access. The target assertion at l_0 is a generic degree two template. The simplification and factorization technique was sufficient to eliminate all the quantifiers. The final invariants obtained are

$$rc_1 + wc_2 + k = k_0 \wedge rw = 0$$

The former accounts for the tokens during the run of the program, while the latter establishes mutual exclusion between the readers and the writers.

LCM-GCD Algorithm

$$\textbf{integer } x_1, x_2, y_1, y_2, y_3, y_4 \textbf{ where } \begin{pmatrix} y_1 = x_1 \wedge \\ y_2 = y_3 = x_2 \wedge \\ y_4 = 0 \end{pmatrix}$$
$$\begin{array}{l} l_0: \textbf{while } (y_1 \neq y_2) \textbf{ do} \\ \quad \left[\begin{array}{l} l_1: \textbf{while } (y_1 > y_2) \textbf{ do} \\ \quad l_1^a: (y_1, y_4) := (y_1 - y_2, y_4 + y_3) \\ l_2: \textbf{while } (y_2 > y_1) \textbf{ do} \\ \quad l_2^a: (y_2, y_3) := (y_2 - y_1, y_3 + y_4) \end{array} \right] \\ \{y_1 = \text{GCD}(x_1, x_2),\ y_3 + y_4 = \text{LCM}(x_1, x_2)\} \end{array}$$

Figure 3. Simultaneous LCM-GCD algorithm

Figure 3 shows a program that calculates the lcm and gcd of integers x_1 and x_2. The integers are modeled as reals for our purpose and the loop conditions at the head of the while loops are abstracted to non-deterministic choices since they lie outside the domain of algebraic assertions. This leads to a transition relation with two loops around the single location l_0. The target invariant for location l_0 is a generic degree two template. The application of the technique produced constraints that resolved completely on factorization and simplification. The resulting invariant obtained at l_0 is $y_1y_3 + y_2y_4 = x_1x_2$. Applying the exit condition $y_1 = y_2$ yields, $y_1(y_3 + y_4) = x_1x_2$. Assuming that $y_1 = \text{GCD}(x_1, x_2)$ and $y_3 + y_4 = \text{LCM}(x_1, x_2)$, the invariant states that

$$\text{LCM}(x_1, x_2) \cdot \text{GCD}(x_1, x_2) = x_1x_2$$

Note that correctness cannot be inferred directly since LCM and GCD functions cannot be expressed algebraically.

$$\textbf{real } y_1, y_2, y_3, y_4, x_1, x_2 \textbf{ where} \begin{pmatrix} y_1 = x_1 \wedge y_2 = x_2 \\ y_3 = 1 \wedge y_4 = 0 \end{pmatrix}$$
$$\begin{array}{l} l_0: \textbf{while } (y_1 \geq y_2) \textbf{ do} \\ \quad l_0^a: (y_2, y_3) := (2y_2, 2y_3) \\ l_1: \textbf{while } (true) \textbf{ do} \\ \quad \left[\begin{array}{ll} l_1^a: & \textbf{if } (y_1 \geq y_2) \textbf{ then} \\ & \quad (y_1, y_4) := (y_1 - y_2, y_4 + y_3) \\ l_1^b: & \textbf{if } (y_3 = 1) \textbf{ then} \\ & \textbf{return } (q = y_4,\ r = y_1) \\ l_1^c: & (y_2, y_3) := (\frac{y_2}{2}, \frac{y_3}{2}) \end{array} \right] \end{array}$$

Figure 4. Hardware Style Division Algorithm

Hardware Style Division Algorithm

Figure 4 shows a procedure, taken from [19], to divide two numbers x_1 and x_2, which we model as reals, in order to apply our technique. The branch conditions that involve inequalities are modeled as non-deterministic choices. The cutpoints used are l_0 and l_1. The target invariant at locations l_0 and l_1 were degree two templates, yielding a total of 56 template variables. Expressing the program as a transition system, we obtain four transitions, one self-loop around l_0, two loops around l_1 and one transition from l_0 to l_1. The constraints generated were simplified using factorization and linearizations. All but two of the cases in the result were linear. The remaining two cases contained simple two variable quantifier elimination instances that were resolved by hand. The following invariants were obtained at l_0 and l_1:

$$\begin{array}{ll} \eta(l_0): y_1 = x_1 & \eta(l_1): true \\ \eta(l_0): y_4 = 0 & \eta(l_1): true \\ \eta(l_0): \varphi_1 \equiv y_2 - y_3x_2 = 0 & \eta(l_1): \varphi_1 \equiv y_2 - y_3x_2 = 0 \end{array}$$

On strengthening the transition relations with the invariants above, and repeating the process, we obtained additional invariant $\varphi_2: y_1y_3 + y_2y_4 - y_3x_1 = 0$ at l_0, l_1.

The loop exits when $y_3 = 1$, $q = y_4$, $r = y_1$. Substituting this on the invariants obtained at l_1, we obtain that $\varphi_2: x_1 = r + qy_2$ and $y_2 = x_2$. Therefore, we can infer $x_1 = r + x_2q$ at the loop exit. This shows one of the specifications of the division algorithm, the other being $y_1 < x_1$, which lies outside the domain of algebraic invariants. Note that some additional invariants are required to justify the fact that the division in l_1^c is applied to integers that are always even. These invariants are, however, not expressible in our assertion language.

6 Conclusion

In this paper, we presented a reduction from the algebraic invariant generation problem for algebraic transition systems to a parametric linear constraint solving problem, so that any solution to the constraints corresponds to an inductive assertion. The technique has many advantages; first of all, the degree of the desired invariant does not affect the constraint problem. Secondly, the constraint solving problem can almost always be handled by simple elimination techniques. With a good constraint handling strategy, we are confident that the technique will scale to larger examples. On the other hand, a drawback of the technique is its lack of completeness arising from the restrictions placed on the consecution condition. While many tried and tested methods in the field of static analysis and program verification do not insist on completeness, the lack of completeness can sometimes cause these methods to miss “obvious” invariants for subtle reasons, as has been observed in traditional invariant generation technique using widening [9]. We are working on identifying useful classes of systems where restricted notions of consecution suffice. Currently inequalities are handled as non-deterministic choices. We are developing modifications of the technique that are able to reason with loops involving inequalities.

The technique also requires that the degree bounds on the target assertion be known a priori. This is a drawback for some applications and effective strategies on selecting the degree bounds of the invariant need to be studied. We are also working on comparisons with related approaches like [22] that may yield techniques to eliminate some of these drawbacks.

Acknowledgements: We would like to thank the anonymous reviewers for their detailed comments on an earlier version of this paper.

7 References

[1] BAADER, F., AND NIPKOW, T. *Term Rewriting and* All That. Cambridge University Press, 1998.

[2] BALLARIN, C., AND KAUERS, M. Solving parametric linear systems: an experiment with constraint algebraic programming. In *Eighth Rhine Workshop on Computer Algebra* (2002), pp. 101–114.

[3] BENSALEM, S., BOZGA, M., FERNANDEZ, J.-C., GHIRVU, L., AND LAKHNECH, Y. A transformational approach for generating non-linear invariants. In *Static Analysis Symposium* (June 2000), vol. 1824 of *LNCS*, Springer Verlag.

[4] BENSALEM, S., LAKHNECH, Y., AND SAIDI, H. Powerful techniques for the automatic generation of invariants. In *Computer-Aided Verification* (1996), vol. 1102 of *LNCS*, pp. 323–335.

[5] BJØRNER, N. S., BROWNE, A., AND MANNA, Z. Automatic generation of invariants and intermediate assertions. *Theoretical Comput. Sci. 173*, 1 (Feb. 1997), 49–87.

[6] BULTAN, T., GERBER, R., AND PUGH, W. Symbolic model checking of infinite state systems using Presburger arithmetic. In *Computer-Aided Verification* (June 1997), vol. 1254 of *LNCS*, springer, pp. 400–411.

[7] COLLINS, G. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages* (1975), H.Brakhage, Ed., vol. 33 of *LNCS*, pp. 134–183.

[8] COLLINS, G. E., AND HONG, H. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation 12*, 3 (sep 1991), 299–328.

[9] COLÓN, M., SANKARANARAYANAN, S., AND SIPMA, H. Linear invariant generation using non-linear constraint solving. In *Computer Aided Verification* (July 2003), F. Somenzi and W. H. Jr, Eds., vol. 2725 of *LNCS*, Springer Verlag, pp. 420–433.

[10] 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.

[11] 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.

[12] COX, D., LITTLE, J., AND O'SHEA, D. *Ideals, Varieties and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, 1991.

[13] DOLZMANN, A., AND STURM, T. REDLOG: Computer algebra meets computer logic. *ACM SIGSAM Bulletin 31*, 2 (June 1997), 2–9.

[14] FLOYD, R. W. Assigning meanings to programs. *Proc. Symposia in Applied Mathematics 19* (1967), 19–32.

[15] HENZINGER, T. A., AND HO, P. HYTECH: The Cornell hybrid technology tool. In *Hybrid Systems II* (1995), vol. 999 of *LNCS*, pp. 265–293.

[16] HOARE, C. A. R. An axiomatic basis for computer programming. *Commun. ACM 12*, 10 (1969), 576–580.

[17] JAFFAR, J., AND LASSEZ, J.-L. Constraint logic programming. In *Principles of Programming Languages(*POPL*)* (Jan. 1987), pp. 111–119.

[18] KARR, M. Affine relationships among variables of a program. *Acta Inf. 6* (1976), 133–151.

[19] MANNA, Z. *Mathematical Theory of Computation*. McGraw-Hill, 1974.

[20] MANNA, Z., AND PNUELI, A. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.

[21] MISHRA, B., AND YAP, C. Notes on Gröbner bases. *Information Sciences 48* (1989), 219–252.

[22] MÜLLER-OLM, M., AND SEIDL, H. Polynomial constants are decidable. In *Static Analysis Symposium (SAS 2002)* (2002), vol. 2477 of *LNCS*, pp. 4–19.

[23] SCHRIJVER, A. *Theory of Linear and Integer Programming*. Wiley, 1986.

[24] SIT, W. Y. An algorithm for solving parametric linear systems. *Journal of Symbolic Computation 13*, 3 (April 1992), 353–394.

[25] TARSKI, A. A decision method for elementary algebra and geometry. *Univ. of California Press, Berkeley 5* (1951).

[26] TIWARI, A., RUESS, H., SAÏDI, H., AND SHANKAR, N. A technique for invariant generation. In *TACAS 2001* (2001), vol. 2031 of *LNCS*, Springer-Verlag, pp. 113–127.

[27] WEISPFENNING, V. The complexity of linear problems in fields. *Journal of Symbolic Computation 5*, 1-2 (April 1988), 3–27.

[28] WEISPFENNING, V. Comprehensive Gröbner bases. *Journal of Symbolic Computation 14* (1992), 1–29.

[29] WEISPFENNING, V. Quantifier elimination for real algebra—the quadratic case and beyond. In *Applied Algebra and Error-Correcting Codes (AAECC) 8* (1997), pp. 85–101.

[30] WINDSTEIGER, W., AND BUCHBERGER, B. Groebner: A library for computing grobner bases based on saclib. Tech. rep., RISC-Linz, 1993.

8 Appendix

In this appendix, we shall prove a few useful theorems mentioned in other results along with the confluence of the Gröbner basis reduction on templates. The key idea is to reduce any failure of confluence on templates to a failure of confluence on the instantiation of the template under an appropriate environment. Two templates are called *identical* if the coefficient corresponding to a term t in one template is the same as the coefficient corresponding to t in the other.

Claim 1 *Let f, f' be templates and α be any environment.*

1. $\alpha(f+g) = \alpha(f) + \alpha(g)$,
2. $c \cdot \alpha(f) = \alpha(c \cdot f)$, *for any* $c \in \Re$.

Theorem 8 (Consistency) *Let $f \xrightarrow{p} f'$ for templates f, f' over template variables in A. Then, for an arbitrary A-environment α, $\alpha(f) \xrightarrow{p} \alpha(f')$ or $\alpha(f) = \alpha(f')$. Conversely, if for some α, $\alpha(f) \xrightarrow{p} h$ then there is a f' such that $h = \alpha(f')$ and $f \xrightarrow{p} f'$.*

PROOF. We have that $f' = f - \frac{c \cdot t}{\mathrm{LT}(p)} p$ for some term $c \cdot t$ in f wherein $c(a_0, \ldots, a_m)$ is a linear expression. If $\alpha(c) = 0$ then $\alpha(f) = \alpha(f')$, or else, if $\alpha(c) \neq 0$ then we have that $\alpha(f') = \alpha(f) - \frac{\alpha(c \cdot t)}{\mathrm{LT}(p)} p$. In this case, $\alpha(f) \xrightarrow{p} \alpha(f')$.

On the other hand, let $\alpha(f) \xrightarrow{p} h$, for some template f and polynomial p. Let $\alpha(c) \cdot t$ be the term in $\alpha(f)$ that $\mathrm{LT}(p)$ divides. Hence, the result of the reduction is,

$$h = \alpha(f) - \frac{\alpha(c)t}{\mathrm{LT}(p)} p = \alpha(f - \frac{ct}{\mathrm{LT}(p)} p)$$

Therefore, setting $f' = f - \frac{ct}{\mathrm{LT}(p)} p$, we have $\alpha(f') = h$ and $f \xrightarrow{p} f'$. □

Theorem 9 (Template Identity) *Two templates f_1, f_2 over template variables A are not identical iff there is an environment α such that $\alpha(f_1) \not\equiv \alpha(f_2)$.*

PROOF. Since $f_1 \not\equiv f_2$, we have that $f_1 - f_2$ is a non-zero template. Let t be a term in $f_1 - f_2$, with a non-zero coefficient expression c. Let α be an environment s.t. $\alpha(c) \neq 0$. Hence $\alpha(f_1 - f_2) \not\equiv 0$. Thus we have that $\alpha(f_1) \not\equiv \alpha(f_2)$.

The other direction is immediate from the definition of identical templates. □

Theorem 10 (Normal Form Theorem) *A template f is a normal form under $\xrightarrow{G}$ iff for each environment α, $\alpha(f)$ is a normal form under $\xrightarrow{G}$.*

PROOF. To prove the forward implication, assume that f is a normal form under $\xrightarrow{G}$. However assume that $\alpha(f) \xrightarrow{G} h$ for some α. By the reverse direction of the consistency theorem (Theorem 8), we have that there exists f' such that $f \xrightarrow{G} f'$ and $\alpha(f') = h$. This contradicts our assumption that f is in normal form.

To prove the reverse implication, let us assume that f is not in normal form. Then there is a reduction $f \xrightarrow{G} f'$. Let t be the term in f that is replaced by the reduction and c be its non-zero coefficient. By Theorem 8, we have that for each environment α, $\alpha(f) = \alpha(f')$ or $\alpha(f) \xrightarrow{G} \alpha(f')$. We find an environment α such that $\alpha(c) \neq 0$. For such an environment $\alpha(f) \xrightarrow{G} \alpha(f')$, since $\alpha(c) \neq 0$. Thus $\alpha(f)$ is not in normal form w.r.t. $\xrightarrow{G}$. □

Theorem 11 (Confluence of Templates) *Let G be a Gröbner basis and f be a template. Let $f \overset{G}{\twoheadrightarrow} f_1$ and $f \overset{G}{\twoheadrightarrow} f_2$, where f_1, f_2 are normal forms. We have that $f_1 \equiv f_2$ and hence $\xrightarrow{G}$ is confluent for templates.*

PROOF. Assuming otherwise, i.e., $f_1 \not\equiv f_2$, we have by Theorem 9 that $\alpha(f_1) \not\equiv \alpha(f_2)$ for some α. Furthermore, Theorem 10 implies that $\alpha(f_1)$ and $\alpha(f_2)$ are in normal forms. By the forward direction of Theorem 8, we have that $\alpha(f) \overset{G}{\twoheadrightarrow} \alpha(f_i)$, $i = 1, 2$. Hence, by the confluence of the Gröbner basis reduction over real polynomials, we have that $\alpha(f_1) = \alpha(f_2)$, thus leading to a contradiction. □

Theorem 12 (Normal Forms for Templates) *Let f be a template over variables A. Let G be the Gröbner basis of $I = (G)$. Then, for any A-environment α,*

$$\alpha(\mathrm{NF}_G(f)) = \mathrm{NF}_G(\alpha(f))$$

PROOF. The proof is by induction over the length of the minimal sequence of reductions from f to $f' = \mathrm{NF}(f)$. For zero length derivations, $f = \mathrm{NF}(f)$, we have that $\alpha(f)$ is a normal form. Hence, $\alpha(\mathrm{NF}(f)) = \alpha(f) = \mathrm{NF}(\alpha(f))$.

Assuming that the theorem holds for templates which have a length n or less derivation to their normal forms, let $f \xrightarrow{G} f_1 \overset{G}{\twoheadrightarrow}_n \mathrm{NF}(f)$. Hence, by Theorem 8, we have that $\alpha(f) = \alpha(f_1)$ or $\alpha(f) \xrightarrow{G} \alpha(f_1)$. In either case $\mathrm{NF}(\alpha(f_1)) = \mathrm{NF}(\alpha(f))$. Applying the induction to f_1, we have that $\alpha(\mathrm{NF}(f_1)) = \mathrm{NF}(\alpha(f_1))$. Hence

$$\alpha(\mathrm{NF}(f)) = \alpha(\mathrm{NF}(f_1)) = \mathrm{NF}(\alpha(f_1)) = \mathrm{NF}(\alpha(f))$$

□

Theorem 13 (Template Membership) *Let f be a template and G be a Gröbner basis, such that $I = ((G))$. Let $f' = \mathrm{NF}_G(f)$. For each environment α, $\alpha(f) \in I$ iff $\alpha(f')$ is* identically zero.

PROOF. Given f, G such that $f' = \mathrm{NF}_G(f)$, for any environment α, we have that $\alpha(f') = \alpha(\mathrm{NF}(f)) = \mathrm{NF}(\alpha(f))$. Hence, if $\alpha(f')$ is identically zero, then $\mathrm{NF}(\alpha(f))$ is identically zero, and therefore, $\alpha(f) \in (G)$.

Let $\alpha(f) \in (G)$, hence $\mathrm{NF}_G(\alpha(f))$ is identically zero. However, $\alpha(f') = \alpha(\mathrm{NF}(f)) = \mathrm{NF}(\alpha(f)) \equiv 0$, and hence, $\alpha(\mathrm{NF}(f)) = \alpha(f')$ is identically zero. □