Generating Polynomial Invariants with DISCOVERER and QEPCAD*

Yinghua Chen[1], Bican Xia[1], Lu Yang[2], and Naijun Zhan[3,**]

[1] LMAM & School of Mathematical Sciences, Peking University
[2] Institute of Theoretical Computing, East China Normal University
[3] Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences
South Fourth Street, No. 4, Zhong Guan Cun, Beijing, 100080, P.R. China
znj@ios.ac.cn

Dedicated to Prof. Chaochen Zhou on his 70th Birthday

Abstract. This paper investigates how to apply the techniques on solving semi-algebraic systems to invariant generation of polynomial programs. By our approach, the generated invariants represented as a semi-algebraic system are more expressive than those generated with the well-established approaches in the literature, which are normally represented as a conjunction of polynomial equations. We implement this approach with the computer algebra tools DISCOVERER and QEPCAD[1]. We also explain, through the complexity analysis, why our approach is more efficient and practical than the one of [17] which directly applies first-order quantifier elimination.

Keywords: Program Verification, Invariant Generation, Polynomial Programs, Semi-Algebraic Systems; Quantifier Elimination, DISCOVERER, QEPCAD.

1 Introduction

Loop invariant generation together with loop termination analysis of programs plays a central role in program verification. Since the late sixties (or early seventies) of the 20th century when the so-called Floyd-Hoare-Dijkstra inductive assertion method, the dominant method on automatic verification of programs, [11,14,9] was invented, there have been lots of attempts to handle the loop problems, e.g. [25,13,16,15], but only with a limited success.

Recently, due to the advance of computer algebra, several methods based on symbolic computation have been applied successfully to invariant generation, for example the techniques based on abstract interpretation [7,1,21,6], quantifier elimination [5,17] and polynomial algebra [19,20,22,23,24].

The basic idea behind the abstract interpretation approaches is to perform an approximate symbolic execution of a program until an assertion is reached

* This work is supported in part by NKBRPC-2002cb312200, NKBRPC-2004CB318003, NSFC-60493200, NSFC-60421001, NSFC-60573007, and NKBRPC-2005CB321902.

** Corresponding author.

[1] http://www.cs.usna.edu/~qepcad/B/QEPCAD.html

C.B. Jones, Z. Liu, J. Woodcock (Eds.): Bjorner/Zhou Festschrift, LNCS 4700, pp. 67–82, 2007.

that remain unchanged by further executions of the program. However, in order to guarantee termination, the method introduces imprecision by use of an extrapolation operator called *widening/narrowing*. This operator often causes the technique to produce weak invariants. Moreover, proposing widening/narrowing operators with certain concerns of completeness is not easy and becomes a key challenge for abstract interpretation based techniques [7,1].

In contrast, [19,20,22,23,24] exploited the theory of polynomial algebra to discover invariants of polynomial programs. [19] applied the technique of linear algebra to generate polynomial equations of bounded degree as invariants of programs with affine assignments. [22,23] first proved that the set of polynomials serving as loop invariants has the algebraic structure of an ideal, then proposed an invariant generation algorithm by using fixpoint computation, and finally implemented the algorithm by the Gröbner bases and the elimination theory. The approach is theoretically sound and complete in the sense that if there is an invariant of the loop that can be expressed as a conjunction of polynomial equations, applying the approach can indeed generate it. [24] presented a similar approach to finding polynomial equation invariants whose form is priori determined (called templates) by using an extended Gröbner basis algorithm over templates.

Compared with the polynomial algebraic approaches that can only generate invariants represented as polynomial equations, [5] proposed an approach to generate linear inequalities as invariants for linear programs, based on *Farkas' Lemma* and non-linear constraint solving. In addition, [17] proposed a very general approach for automatic generation of more expressive invariants by exploiting the technique of quantifier elimination, and applied the approach to Presburger Arithmetic and quantifier-free theory of conjunctively closed polynomial equations. Theoretically speaking, the approach can also be applied to the theory of real closed fields, but [17] pointed out that this is impractical in reality because of the high complexity of quantifier elimination, which is double exponential [8]. To handle the problem, [6] exploited the techniques of parametric abstraction, Lagrangian relaxation and semidefinite programming to generate invariants as well as ranking functions of polynomial programs. Compared with the approach of [17], [6]'s is more efficient, as first-order quantifier elimination is not directly applied there. However, [6]'s approach is incomplete in the sense that, for some program that may have ranking functions and invariants of the predefined form, applying the approach may not be able to find them, as Lagrangian relaxation and over-approximation of the positive semi-definiteness of a polynomial are used.

In this paper, we attack the problem raised in [17] on how to efficiently generate polynomial invariants of programs over real closed fields and present a more practical and efficient approach to it by exploiting our results on solving semi-algebraic systems (SASs). The outline of our approach is as follows: we first reduce polynomial invariant generation problem to solving semi-algebraic systems; then apply our theories and tools on solving SASs, in particular, on root classification of parametric SASs [30,31,32] and real root isolation of constant

SASs [28,29], to produce some necessary and sufficient conditions; and finally utilize the technique of quantifier elimination to handle the derived conditions and obtain invariants with the predefined form.

Suppose an SAS S has s (> 0) polynomial equations and m inequations and inequalities. All polynomials are in $n = t + (n - t)$ indeterminates (i.e., $u_1, \ldots, u_t$, $x_1, \ldots, x_{n-t}$) and of degree at most d where t is the dimension of the ideal generated by the s equations. According to [3,8], directly applying the technique of quantifier elimination of real closed fields, the cost for solving S is doubly exponential w.r.t. n. But using our approach, the cost for the first step is almost nothing, and the second step to apply root classification and isolation costs singly exponential w.r.t. n plus doubly exponential w.r.t. t. The cost for the last step is also doubly exponential w.r.t. t. Therefore, as $t < n$, our approach is more efficient than the approaches directly based on the techniques of quantifier elimination and Gröbner basis, such as [17,24], in particular, when t is much less than n. Moreover, our approach is still complete in the sense that whenever there exist invariants of the predefined form, applying our approach can indeed synthesize them, while [6]'s [2] is incomplete. On the other hand, similarly to [17,6], invariants generated by our approach are more expressive, while applying the approaches based on polynomial algebra can only produce conjunction of polynomial equations as invariants.

The rest of this paper is organized as: Section 2 provides a brief review of the theories and tools on solving SASs; Section 3 defines some basic notions, including semi-algebraic transition systems, polynomial programs, invariants, inductive properties and so on; Section 4 is devoted to illustrating our approach in detail with a running example; We provide the complexity analysis of our approach in Section 5; In Section 6, we compare the application of this approach to invariant generating with the one to ranking function discovering; and Section 7 concludes the paper and discusses the future work in this direction.

2 Preliminaries: Theories on Semi-algebraic Systems

In this section, we introduce the cornerstone of our technique, i.e. theories and tools on solving SASs, mainly the theories on root classification of parametric SASs and the tool DISCOVERER.

2.1 Basic Notions

Let $\mathcal{K}[x_1, ..., x_n]$ be the ring of polynomials in n indeterminates, $X = \{x_1, \cdots, x_n\}$, with coefficients in the field $\mathcal{K}$. Let the variables be ordered as $x_1 \prec x_2 \prec \cdots \prec x_n$. Then, the *leading variable* (or *main variable*) of a polynomial p is the variable with the biggest index which indeed occurs in p. If the leading variable of a polynomial p is x_k, p can be collected w.r.t. its leading variable as $p = c_m x_k^m + \cdots + c_0$

[2] As far as efficiency is concerned, we believe that our approach could be at least as good as [6]'s, as the complexity of the semi-definite programming adopted in [6] is also very high.

where m is the *degree* of p w.r.t. x_k and c_is are polynomials in $\mathcal{K}[x_1, ..., x_{k-1}]$. We call $c_m x_k^m$ the *leading term* of p w.r.t. x_k and c_m the *leading coefficient*. For example, let $p(x_1, \ldots, x_5) = x_2^5 + x_3^4 x_4^2 + (2x_2 + x_1)x_4^3$, so, its leading variable, term and coefficient are x_4, $(2x_2 + x_1)x_4^3$ and $2x_2 + x_1$, respectively.

An *atomic polynomial formula* over $\mathcal{K}[x_1, ..., x_n]$ is of the form $p(x_1, \ldots, x_n) \rhd 0$, where $\rhd \in \{=, >, \geq, \neq\}$, while a *polynomial formula* over $\mathcal{K}[x_1, ..., x_n]$ is constructed from atomic polynomial formulae by applying the logical connectives. Conjunctive polynomial formulae are those that are built from atomic polynomial formulae with the logical operator $\wedge$. We will denote by $PF(\{x_1, \ldots, x_n\})$ the set of polynomial formulae and by $CPF(\{x_1, \ldots, x_n\})$ the set of conjunctive polynomial formulae, respectively.

In what follows, we will use $\mathbb{Q}$ to stand for rationales and $\mathbb{R}$ for reals, and fix $\mathcal{K}$ to be $\mathbb{Q}$. In fact, all results discussed below can be applied to $\mathbb{R}$.

In the following, the n indeterminates are divided into two groups: $\mathbf{u} = (u_1, ..., u_t)$ and $\mathbf{x} = (x_1, ..., x_s)$, which are called parameters and variables, respectively, and we sometimes use "," to denote the conjunction of atomic formulae for simplicity.

Definition 1. *A semi-algebraic system is a conjunctive polynomial formula of the following form:*

$$\begin{cases} p_1(\mathbf{u}, \mathbf{x}) = 0, ..., p_r(\mathbf{u}, \mathbf{x}) = 0, \\ g_1(\mathbf{u}, \mathbf{x}) \geq 0, ..., g_k(\mathbf{u}, \mathbf{x}) \geq 0, \\ g_{k+1}(\mathbf{u}, \mathbf{x}) > 0, ..., g_l(\mathbf{u}, \mathbf{x}) > 0, \\ h_1(\mathbf{u}, \mathbf{x}) \neq 0, ..., h_m(\mathbf{u}, \mathbf{x}) \neq 0, \end{cases} \tag{1}$$

where $r > 1, l \geq k \geq 0, m \geq 0$ and all p_i's, g_i's and h_i's are in $\mathbb{Q}[\mathbf{u}, \mathbf{x}] \setminus \mathbb{Q}$. An SAS of the form (1) is called parametric *if $t \neq 0$, otherwise* constant.

An SAS of the form (1) is usually denoted by a quadruple $[\mathbb{P}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{H}]$, where $\mathbb{P} = [p_1, ..., p_r], \mathbb{G}_1 = [g_1, ..., g_k], \mathbb{G}_2 = [g_{k+1}, ..., g_l]$ and $\mathbb{H} = [h_1, ..., h_m]$.

For a constant SAS S, interesting questions are how to compute the number of real solutions of S, and if the number is finite, how to compute these real solutions. For a parametric SAS, the interesting problem is so-called *real solution classification*, that is to determine the condition on the parameters such that the system has the prescribed number of distinct real solutions, possibly infinite.

2.2 Theories on Real Solution Classification

In this subsection, we outline the theories for real root classification of parametric SASs. For details, please be referred to [31,27].

For an SAS S of the form (1), the algorithm for real root classification consists of three main steps. Firstly, transform the equations of S into some sets of equations in triangular form. A set of equations $\mathbb{T} : [T_1, ..., T_k]$ is said to be in triangular form (or a *triangular set*) if the main variable of T_i is less in the order than that of T_j if $i < j$. Roughly speaking, if we rename the variables, a triangular set looks like

$$
\begin{aligned}
&T_1 = T_1(\mathbf{v}, y_1),\\
&T_2 = T_2(\mathbf{v}, y_1, y_2),\\
&\cdots\cdots\\
&T_k = T_k(\mathbf{v}, y_1, \cdots, y_k),
\end{aligned}
$$

where $\mathbf{v}$ are the indeterminates other than y_i. It is obvious that we now only need to consider triangular systems in the following form

$$
\begin{cases}
f_1(\mathbf{u}, x_1) = 0,\\
\quad\vdots\\
f_s(\mathbf{u}, x_1, ..., x_s) = 0,\\
\mathbb{G}_1,\ \mathbb{G}_2,\ \mathbb{H}.
\end{cases}
\tag{2}
$$

Certainly, it can be proven that there exists a correspondence between the solutions of these triangular sets and S's so that we only need to consider the solutions of these triangular sets in order to deal with S's.

Example 1. Consider an SAS $S : [\mathbb{P}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{H}]$ in $\mathbb{Q}[b, x, y, z]$ with $\mathbb{P} = [p_1, p_2, p_3]$, $\mathbb{G}_1 = \emptyset, \mathbb{G}_2 = [x, y, z, b, 2 - b], \mathbb{H} = \emptyset$, where

$$p_1 = x^2 + y^2 - z^2,\ p_2 = (1 - x)^2 - z^2 + 1,\ p_3 = (1 - y)^2 - b^2 z^2 + 1.$$

The equations $\mathbb{P}$ can be decomposed into two triangular sets in $\mathbb{Q}(b)[x, y, z]$

$\mathbb{T}_1 : [b^4 x^2 - 2b^2(b^2 - 2)x + 2b^4 - 8b^2 + 4, -b^2 y + b^2 x + 2 - 2b^2, b^4 z^2 + 4b^2 x - 8b^2 + 4]$,
$\mathbb{T}_2 : [x^2 - 2x + 2, y + x - 2, z]$,

with the relation

$$\text{Zero}(\mathbb{P}) = \text{Zero}(\mathbb{T}_1/b) \bigcup \text{Zero}(\mathbb{T}_2)$$

where Zero() means the set of zeros and $\text{Zero}(\mathbb{T}_1/b) = \text{Zero}(\mathbb{T}_1) \setminus \text{Zero}(b)$.

Second, compute a so-called *border polynomial* from the resulting triangular systems, say $[\mathbb{T}_i, \mathbb{G}_1, \mathbb{G}_2, \mathbb{H}]$. We need to introduce some concepts. Suppose F and G are polynomials in x with degrees m and l, respectively. Thus, they can be written in the following forms

$$F = a_0 x^m + a_1 x^{m-1} + \cdots + a_{m-1} x + a_m,\ G = b_0 x^l + b_1 x^{l-1} + \cdots + b_{l-1} x + b_l.$$

The following $(m + l) \times (m + l)$ matrix (those entries except a_i, b_j are all zero)

$$
\begin{pmatrix}
a_0 & a_1 & \cdots & a_m & & & \\
 & a_0 & a_1 & \cdots & a_m & & \\
 & & \ddots & \ddots & & \ddots & \\
 & & & a_0 & a_1 & \cdots & a_m \\
b_0 & b_1 & \cdots & b_l & & & \\
 & b_0 & b_1 & \cdots & b_l & & \\
 & & \ddots & \ddots & & \ddots & \\
 & & & b_0 & b_1 & \cdots & b_l
\end{pmatrix}
\begin{matrix}
\left.\begin{matrix}\\\\\\\end{matrix}\right\} l\\
\left.\begin{matrix}\\\\\\\end{matrix}\right\} m
\end{matrix}
,
$$

is called the *Sylvester matrix* of F and G w.r.t. x. The determinant of the matrix is called the *Sylvester resultant* or *resultant* of F and G w.r.t. x and is denoted by $\mathrm{res}(F, G, x)$.

For system (2), we compute the resultant of f_s and f_s' *w.r.t.* x_s and denote it by $\mathrm{dis}(f_s)$ (it has the leading coefficient and discriminant of f_s as factors). Then we compute the *successive resultant* of $\mathrm{dis}(f_s)$ and the triangular set $\{f_{s-1}, ..., f_1\}$. That is, we compute $\mathrm{res}(\mathrm{res}(\cdots \mathrm{res}(\mathrm{res}(\mathrm{dis}(f_s), f_{s-1}, x_{s-1}), f_{s-2}, x_{s-2}) \cdots), f_1, x_1)$ and denote it by $\mathrm{res}(\mathrm{dis}(f_s); f_{s-1}, ..., f_1)$ or simply R_s. Similarly, for each i $(1 < i \leq s)$, we compute $R_i = \mathrm{res}(\mathrm{dis}(f_i); f_{i-1}, ..., f_1)$ and $R_1 = \mathrm{dis}(f_1)$.

For each of those inequalities and inequations, we compute the successive resultant of g_j (or h_j) w.r.t. the triangular set $[f_1, ..., f_s]$ and denote it by Q_j (resp. Q_{l+j}).

Definition 2. *For an SAS T as defined by (2), the* border polynomial *of T is*

$$BP = \prod_{i=1}^{s} R_i \prod_{j=1}^{l+m} Q_j.$$

Sometimes, for brevity, we also abuse BP to denote the square-free part or the set of square-free factors of BP.

Example 2. For the system S in Example 1, the border polynomial is

$$BP = b(b-2)(b+2)(b^2-2)(b^4-4b^2+2)(2b^4-2b^2+1).$$

From the result in [31,27], we may assume $BP \neq 0$. In fact, if any factor of BP is a zero polynomial, we can further decompose the system into new systems with such a property. For a parametric SAS, its border polynomial is a polynomial in the parameters with the following property.

Theorem 1. *Suppose S is a parametric SAS as defined by (2) and BP its border polynomial. Then, in each connected component of the complement of $BP = 0$ in parametric space $\mathbb{R}^d$, the number of distinct real solutions of S is constant.*

Third, $BP = 0$ decomposes the parametric space into a finite number of connected region. We then choose sample points in each connected component of the complement of $BP = 0$ and compute the number of distinct real solutions of S at each sample point. Note that sample points can be obtained by the partial cylindrical algebra decomposition (PCAD) algorithm [4].

Example 3. For the system S in Example 1, $BP = 0$ gives

$$b = 0, \ \pm 2, \ \pm\sqrt{2}, \ \pm\sqrt{2 \pm \sqrt{2}}.$$

The reals are divided into ten open intervals by these points. Because $0 < b < 2$, we only need to choose one point, for example, $\frac{1}{2}, 1, \frac{3}{2}, \frac{15}{8}$, from each of the four intervals contained in $(0, 2)$, respectively. Then, we substitute each of the four values for b in the system, and compute the number of distinct real solutions of the system, consequently obtain the system has respectively $0, 1, 0$ and 0 distinct real solutions.

The above three steps constitute the main part of the algorithm in [31,34,27], which, for any input SAS S, outputs the so-called border polynomial BP and a quantifier-free formula Ψ in terms of polynomials in parameters $\mathbf{u}$ (and possible some variables) such that, provided $BP \neq 0$, Ψ is the necessary and sufficient condition for S to have the given number (possibly infinite) of real solutions. Since BP is a polynomial in parameters, $BP = 0$ can be viewed as a degenerated condition. Therefore, the outputs of the above three steps can be read as "if $BP \neq 0$, the necessary and sufficient condition for S to have the given number (possibly infinite) of real solutions is Ψ."

Remark 1. If we want to discuss the case when parameters degenerate, i.e., $BP = 0$, we put $BP = 0$ (or some of its factors) into the system and apply a similar procedure to handle the new SAS.

Example 4. By the steps described above, we obtain the necessary and sufficient condition for S to have one distinct real solution is $b^2 - 2 < 0 \ \wedge \ b^4 - 4b^2 + 2 < 0$ provided $BP \neq 0$. Now, if $b^2 - 2 = 0$, adding the equation into the system, we obtain a new SAS: $[\ [b^2 - 2, p_1, p_2, p_3], [\ ], \mathbb{G}_2, [\ ]\ ]$. By the algorithm in [28,29], we know the system has no real solutions.

2.3 A Computer Algebra Tool: DISCOVERER

We have implemented the above algorithm and some other algorithms in Maple as a computer algebra tool, named DISCOVERER. The reader can download the tool for free via "`http://www.is.pku.edu.cn/~xbc/discoverer.html`". The prerequisite to run the package is Maple 7.0 or a later version of it.

The main features of DISCOVERER include

Real Solution Classification of Parametric Semi-algebraic Systems

For a parametric SAS T of the form (1) and an argument N, where N is one of the following three forms:

- a non-negative integer b;
- a range $b..c$, where b, c are non-negative integers and $b < c$;
- a range $b..w$, where b is a non-negative integer and w is a name without value, standing for $+\infty$,

DISCOVERER can determine the conditions on $\mathbf{u}$ such that the number of the distinct real solutions of T equals to N if N is an integer, otherwise falls in the scope N. This is by calling

$$\mathbf{tofind}([\mathbb{P}], [\mathbb{G}_1], [\mathbb{G}_2], [\mathbb{H}], [x_1, ..., x_s], [u_1, ..., u_t], N),$$

and results in the necessary and sufficient condition as well as the *border polynomial* BP of T in u such that the number of the distinct real solutions of T exactly equals to N or belongs to N provided $BP \neq 0$. If T has infinite real solutions for generic value of parameters, BP may have some variables. Then, for the "boundaries" produced by "**tofind**", i.e. $BP = 0$, we can call

$$\mathbf{Tofind}([\mathbb{P}, BP], [\mathbb{G}_1], [\mathbb{G}_2], [\mathbb{H}], [x_1, ..., x_s], [u_1, ..., u_t], N)$$

to obtain some further conditions on the parameters.

Real Solution Isolation of Constant Semi-algebraic Systems

For a constant SAS T (i.e., $t = 0$) of the form (1), if T has only a finite number of real solutions, DISCOVERER can determine the number of distinct real solutions of T, say n, and moreover, can find out n disjoint cubes with rational vertices in each of which there is only one solution. In addition, the width of the cubes can be less than any given positive real. The two functions are realized through calling

$$\textbf{nearsolve}([\mathbb{P}], [\mathbb{G}_1], [\mathbb{G}_2], [\mathbb{H}], [x_1, ..., x_s]) \text{ and}$$
$$\textbf{realzeros}([\mathbb{P}], [\mathbb{G}_1], [\mathbb{G}_2], [\mathbb{H}], [x_1, ..., x_s], w),$$

respectively, where w is optional and used to indicate the maximum size of the output cubes.

3 Semi-algebraic Transition Systems and Invariants

In this section, we extend the notion of algebraic transition systems in [24] to semi-algebraic transition systems (SATSs) to represent polynomial programs. An Algebraic Transition System (ATS) is a special case of standard transition system, in which the initial condition and all transitions are specified in terms of polynomial equations; while in an SATS, each transition is equipped with a conjunctive polynomial formula as guard, and its initial and loop conditions possibly contain polynomial inequations and inequalities. It is easy to see that ATS is a special case of SATS. Formally,

Definition 3. *A semi-algebraic transition system is a quintuple $\langle V, L, T, \ell_0, \Theta\rangle$, where V is a set of program variables, L is a set of locations, and T is a set of transitions. Each transition $\tau \in T$ is a quadruple $\langle \ell_1, \ell_2, \rho_\tau, \theta_\tau\rangle$, where ℓ_1 and ℓ_2 are the pre- and post- locations of the transition, $\rho_\tau \in CPF(V, V')$ is the transition relation, and $\theta_\tau \in CPF(V)$ is the guard of the transition. Only if θ_τ holds, the transition can take place. Here, we use V' (variables with prime) to denote the next-state variables. The location ℓ_0 is the initial location, and $\Theta \in CPF(V)$ is the initial condition.*

If a transition τ changes nothing, i.e. $\rho_\tau \equiv \bigwedge_{v\in V} v' = v$, we denote by *skip* ρ_τ. Meanwhile, a transition $\tau = \langle l_1, l_2, \rho_\tau, \theta_\tau\rangle$ is abbreviated as $\langle l_1, l_2, \rho_\tau\rangle$, if θ_τ is *true*.

Note that in the above definition, for simplicity, we require that each guard should be a conjunctive polynomial formula. In fact, we can drop such a restriction, as for any transition with a disjunctive guard we can split it into multiple transitions with the same pre- and post- locations and transition relation, but each of which takes a disjunct of the original guard as its guard.

A state is an evaluation of the variables in V and all states are denoted by $Val(V)$. Without confusion we will use V to denote both the variable set and an arbitrary state, and use $F(V)$ to mean the (truth) value of function (formula) F under the state V. The semantics of SATSs can be explained through state transitions as usual.

A transition is called *separable* if its transition relation is a conjunctive formula of equations which define variables in V' equal to polynomial expressions over

variables in V. It is easy to see that the composition of two separable transitions is equivalent to a single separable one. An SATS is called *separable* if each transition of the system is separable. In a separable system, the composition of transitions along a path of the system is also equivalent to a single separable transition. We will only concentrate on separable SATSs as any polynomial program can easily be represented by a separable SATS (see [18]). Any SATS in the rest of the paper is always assumed separable, unless otherwise stated.

Informally, an *invariant* of a program at a location is an assertion that is *true* under any program state reaching the location. An invariant of a program can be seen as a mapping to map each location to an assertion which has inductive property, that is, *initial* and *consecutive*. *Initial* means that the image of the mapping at the initial location holds on the loop entry, i.e. the invariant of the initial location holds on the loop entry; whereas *consecutive* means that for any transition the invariant at the pre-location together with the transition relation and its guard implies the invariant at the post-location. In many cases, people only consider an invariant at the initial location and do not care about invariants at other locations. In this case, we can assume the invariants at other locations are all *true* and therefore *initial* and *consecutive* mean that the invariant holds on the entry, and is preserved under every cycle back to the initial location.

Definition 4 (Invariant at a Location). *Let $P = \langle V, L, T, l_0, \Theta\rangle$ be an SATS. An invariant at a location $l \in L$ is a conjunctive polynomial formula $\phi \in PF(V)$, such that ϕ holds on all states that can be reached at location l.*

Definition 5 (Invariant of a Program). *An* assertion map *for an SATS $P = \langle V, L, T, l_0, \Theta\rangle$ is a map $\eta : L \mapsto PF(V)$ that associates each location of P with a formula of $PF(V)$. An assertion map of P is said to be an* invariant *of P iff the following conditions hold:*

Initial: $\Theta(V_0) \models \eta(l_0)$.
Consecutive: *For each transition $\tau = \langle l_i, l_j, \rho_\tau, \theta_\tau\rangle$,*

$$\eta(l_i)(V) \wedge \rho_\tau(V, V') \wedge \theta_\tau(V) \models \eta(l_j)(V').$$

4 Polynomial Invariants Generation

Similarly to [24], given an SATS S, we predetermine an invariant as a parametric SAS (PSAS for short) at each of the underlining locations (if no invariant is predefined for a location, it is assumed that the mapping takes *true* as value at the location) and therefore all these predefined PSASs form a parametric invariant of S by the Definitions. Subsequently, according to the initial and consecutive conditions of the mapping, we can obtain a set of PSASs such that the mapping is an invariant of the program iff each element the resulted set has no real solution. Afterwards, we apply the algorithm on root classification of PSASs to each of them and obtain a corresponding necessary and sufficient condition on the parameters of the PSAS such that the PSAS has no real solution. Finally, applying quantifier elimination technique, we can get the instantiations of

these parameters and therefore get an invariant for each underlining location by replacing with the resulted instantiations the parameters of the predetermined parametric PSAS. The above procedure are supported by the computer algebra tools DISCOVERER and QEPCAD.

We will use the following example to demonstrate our approach in details.

Example 5. Consider a program shown in Fig.1 (a).

Integer $(x, y) := (0, 0)$;
l_0 : **while** $x \geq 0 \wedge y \geq 0$ **do**
$(x, y) := (x + y^2, y + 1)$;
end while

(a)

$P = \{$
$V = \{x, y\}$
$L = \{l_0\}$
$T = \{\tau\}$ $\}$
where
$\tau = \langle l_0, l_0, x' - x - y^2 = 0 \wedge y' - y - 1 = 0,$
$x \geq 0 \wedge y \geq 0 \rangle$

(b)

Fig. 1.

Thus, the corresponding SATS can be represented as in Fig.1 (b).

In the following, we concretize the above idea and demonstrate with the toy example.

Predefining Invariant. Predetermine a template of invariants at each of the underlining location, which is a PSAS, i.e. the conjunction of a set of atomic polynomial formula. All of these predefined PSASs form a parametric invariant of the program. For example, we can assume a template of invariants of P at l_0 in Example 5 as

$$eq(x, y) = a_1 y^3 + a_2 y^2 + a_3 x - a_4 y = 0 \tag{3}$$
$$ineq(x, y) = b_1 x + b_2 y^2 + b_3 y + b_4 > 0, \tag{4}$$

where $a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4$ are parameters. Therefore, $\eta(l_0) = (3) \wedge (4)$.
Note that theoretically speaking we can predefine a PSAS as an invariant at each location like in the above example, but this will raise the complexity dramatically as thus the number of parameters is so large (the reader will see this point from the complexity analysis in the later). In practice, alternatively, we will split a complicated invariant to several simple invariants such that the image of every of these simple invariants at each location is just one of the atomic subformulae of the image of the complicated invariant at the location. For example, in the above example, we can split η to η_1 and η_2 by letting $\eta_1(l_0) = (3)$ and $\eta_2(l_0) = (4)$. It is easy to prove that a program has a complicated invariant iff the corresponding simple invariant exist, for instance, η exists iff η_1 and η_2 exist. This is because every invariant of a program is determined by the program itself.

Deriving PSASs from Initial Condition and Solving. According to the initial condition in Definition 5, we have $\Theta \models \eta(l_0)$ which means that each real solution of Θ must satisfy $\eta(l_0)$. In other words, $\Theta \wedge \neg\eta(l_0)$ has no real

solution. This implies that for each atomic polynomial formula ϕ in $\eta(l_0)$, $\Theta \wedge \neg\phi$ has no real solution. Note that $\eta(l_0)$ is the conjunction of a set of atomic polynomials and $\Theta \wedge \neg\phi$ is a PSAS according to the definition. Thus, applying the tool DISCOVERER to the resulted PSAS $\Theta \wedge \neg\phi$, we get a necessary and sufficient condition of the derived PSAS having no real solution. The condition may contain the occurrences of some program variables. In this case, the condition should hold for any instantiations of these variables. Thus, by introducing universal quantifications of these variables (we usually add a scope to each of these variables according to different situations) and then applying QEPCAD, we can get a necessary and sufficient condition only on the presumed parameters.
Repeatedly apply the procedure to each atomic polynomial formula of the predefined invariant at l_0 and then collect all the resulted conditions.

Example 6. In Example 5, $\Theta \models \eta_1(l_0)$ is equivalent to

$$x = 0, y = 0, eq(x, y) \neq 0 \tag{5}$$

has no real solution. By calling

$$\textbf{tofind}(([x, y], [\,], [\,], [eq(x, y)], [x, y], [a_1, a_2, a_3, a_4], 0)$$

we get that (5) has no real solution iff *true*.
Similarly, $\Theta \models \eta_2(l_0)$ is equivalent to

$$x = 0, y = 0, ineq(x, y) \leq 0 \tag{6}$$

has no real solution. Calling

$$\textbf{tofind}([x, y], [-ineq(x, y)], [\,], [\,], [x, y], [b_1, b_2, b_3, b_4], 0)$$

we get that (6) has no real solution iff

$$b_4 > 0. \tag{7}$$

Deriving PSASs from Consecutive Condition and Solving. From Definition 5, for each transition $\tau = \langle l_i, l_j, \rho_\tau, \theta_\tau \rangle$,

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

so $\eta(l_i) \wedge \rho_\tau \wedge \theta_\tau \wedge \neg\eta(l_j)$ has no real solution. This implies that for each atomic polynomial formula ϕ

$$\eta(l_i) \wedge \rho_\tau \wedge \theta_\tau \wedge \neg\phi \tag{8}$$

has no real solution. It is easy to see that (8) is a PSAS according to Definition 1, so applying the tool DISCOVERER, we obtain a necessary and sufficient condition on the parameters such that (8) has no real solution. Subsequently, similarly to Step 2, we may need to exploit the quantifier elimination tool QEPCAD to reduce the resulted condition in order to get a necessary and sufficient condition only on the presumed parameters.

Example 7. In Example 5, for the invariant η_1, we have

$$eq(x,y)=0\wedge x'-x-y^2=0\wedge y'-y-1=0\models eq(x',y')=0. \tag{9}$$

This is equivalent to

$$eq(x,y)=0\wedge x'-x-y^2=0\wedge y'-y-1=0\wedge eq(x',y')\neq 0 \tag{10}$$

has no real solution. Calling

$$\textbf{tofind}([x'-x-y^2,y'-y-1,eq(x,y)],[\,],[\,],[eq(x',y')\,],\\ [x',y',x],[y,a_1,a_2,a_3,a_4],0),$$

it follows that (10) has no real solution iff

$$a_3y^2+3a_1y^2+2ya_2+3a_1y-a_4+a_2+a_1=0\wedge \tag{11}$$

$$a_3(a_1y^2+ya_2-a_4)\leq 0. \tag{12}$$

Further by Basic Algebraic Theorem and simplifying by QEPCAD, (11) $\wedge$ (12) holds for all y iff

$$-a_4+a_2+a_1=0\wedge 3a_1+2a_2=0\wedge a_3+3a_1=0. \tag{13}$$

Regarding the invariant η_2, we have

$$ineq(x,y)>0\wedge x'-x-y^2=0\wedge y'-y-1=0\models ineq(x',y')>0. \tag{14}$$

This is equivalent to

$$ineq(x,y)>0\wedge x'-x-y^2=0\wedge y'-y-1=0\wedge ineq(x',y')\leq 0 \tag{15}$$

has no real solution. Calling

$$\textbf{tofind}([x'-x-y^2,y'-y-1],[-ineq(x',y')],[ineq(x,y)],[\,],\\ [x',y'],[x,y,b_1,b_2,b_3,b_4],0),$$

it follows that (15) has no real solution iff

$$b_4+b_3+b_2+2b_2y+b_3y+b_2y^2+b_1x+b_1y^2>0. \tag{16}$$

It is easy to see that (16) should hold for all $y\geq 0$, and thus, by applying QEPCAD to eliminate the quantifiers $\forall y\geq 0$ over (16), we get

$$b_1+b_2\geq 0\wedge b_1\geq 0\wedge b_2+b_3+b_4>0\wedge\\ (b_3+2b_2\geq 0\vee(b_1b_2+b_2^2\geq 0\wedge 4b_2b_4+4b_1b_4+4b_1b_3+4b_1b_2-b_3^2>0)) \tag{17}$$

Generating Invariant. According to the results obtained from Steps 1, 2 and 3, we can get the final necessary and sufficient condition only on the parameters of each of the invariant templates. If the condition is too complicated, we can utilize the function of PCAD of DISCOVERER or QEPCAD to prove if or not the condition is satisfied. If yes, the tool can produce the instantiations of these parameters. Thus, we can get an invariant of the predetermined form by replacing the parameters with the instantiations, respectively.

Example 8. From Examples 6 & 7, it follows the necessary and sufficient condition on the parameters of η_1 is (13). By using DISCOVERER, we get an instantiation

$$(a_1, a_2, a_3, a_4) = (-2, 3, 6, 1).$$

Thus, $\eta_1(l_0) = -2y^3 + 3y^2 + 6x - y = 0$. Also, the necessary and sufficient condition on the parameters of η_2 is (7) $\wedge$ (17). By PCAD of DISCOVERER, it results the following instantiation

$$(b_1, b_2, b_3, b_4) = (1, -1, 2, 1)$$

that is, $\eta_2(l_0) = x - y^2 + 2y + 1 > 0$. Totally, we get the following invariant for the program P:

$$\begin{cases} -2y^3 + 3y^2 + 6x - y = 0, \\ x - y^2 + 2y + 1 > 0 \end{cases}$$

Note that the above procedure is *complete* in the sense that for any given predefined parametric invariant, the procedure can always give you an answer, yes or no. Therefore, we can conclude that our approach is also *complete* in the sense that once the given polynomial program has a polynomial invariant, our approach can indeed find it theoretically. This is because we can assume parametric invariants in program variables of different degrees, and repeatedly apply the above procedure until we obtain a polynomial invariant.

5 Complexity Analysis

Assume given an SATS $P = \langle V, L, T, l_0, \Theta \rangle$, applying the above procedure, we obtain k distinct PSASs so that the predefined parametric invariants form an invariant of the program iff none of these k PSASs has any real solution. W.l.o.g., suppose each of these k PSASs has at most s polynomial equations, and m inequations and inequalities. All polynomials are in n indeterminates (i.e., variables and parameters) and of degrees at most d.

For a PSAS S, by [3], CAD (*cylindrical algebraic decomposition*) based quantifier elimination on S has complexity $\mathcal{O}((2d)^{2^{2n+8}}(s+m)^{2^{n+6}})$, which is double exponential w.r.t. n. Thus, the total cost is $\mathcal{O}(k(2d)^{2^{2n+8}}(s+m)^{2^{n+6}})$ for directly applying the technique of quantifier elimination to generate an invariant of a program as advocated by Kapur [17].

In contrast, the cost of our approach includes two parts: one is for applying real solution classification to generate condition on the parameters possibly together with some program variables; the other is for applying first-order quantifier elimination to produce condition only on the parameters (if necessary) and further exploiting PCAD to obtain the instantiations of these parameters.

The first part consists of three main steps. Firstly, we transform the equations in S into triangular sets (i.e., equations in triangular form) by Ritt-Wu's method. By [12], the complexity of computing the first characteristic set is $\mathcal{O}(s^{\mathcal{O}(n)}(d+1)^{\mathcal{O}(n^3)})$. Thus, the complexity of this step is $\mathcal{O}(s^{n^{\mathcal{O}(1)}}(d+1)^{n^{\mathcal{O}(1)}})$, which is usually called a singly exponential complexity w.r.t. n. Secondly, we

compute a *border polynomial* (BP) from the triangularized systems through resultant computation. Because the polynomials in the first computed characteristic sets are of degree $\mathcal{O}(s(d+1)^{\mathcal{O}(n^2)})$ by [12], the polynomials in the computed triangular sets are of degree $\mathcal{O}(s^{n^{\mathcal{O}(1)}}(d+1)^{n^{\mathcal{O}(1)}})$. Thus, the complexity of computing BP is at most $(s+m)\mathcal{O}(s^{3s+sn^{\mathcal{O}(1)}}(d+1)^{sn^{\mathcal{O}(1)}})$ because the complexity of computing the resultant of two polynomials with degree d is at most $\mathcal{O}(d^3)$. Moreover, the degree of BP is at most $D = \mathcal{O}(s^{\mathcal{O}(s^2+s^2n^{\mathcal{O}(1)})}(d+1)^{\mathcal{O}(s^2n^{\mathcal{O}(1)})})$. Finally, we use a partial CAD algorithm with BP to obtain real solution classification. The complexity of this step is at most the complexity of performing quantifier elimination on BP using CAD. Suppose the dimension of the ideal generated by the s polynomial equations is t, then BP has at most t indeterminates. Thus, by [3], the complexity of this step is at most $\mathcal{O}(2D^{2^{2t+8}})$, which is double exponential with respect to t. In a word, the cost for this part is singly exponential in n and doubly exponential in t.

As the biggest degree of polynomials in the generated necessary and sufficient condition from the above is at most D, the cost for the second part is $\mathcal{O}(2D^{2^{2t+8}})$ as well, which is doubly exponential in t, according to [3].

So, compared to directly applying quantifier elimination, our approach can dramatically reduce the complexity, in particular when t is much less than n.

6 Generating Invariants vs. Discovering Ranking Functions

In [2], we showed how to apply the approach to discovering non-linear ranking functions. Although invariants and ranking functions both have inductive properties, the former is inductive w.r.t. a small step, i.e. each of single transitions of the given program in contrast that the latter is inductive w.r.t. a big step, that is each of circle transition at the initial location of the program. The difference results that as far as invariant generation is concerned, our approach can be simply applied to single loop programs as well as nested loop programs, without any change; but regarding the discovery of ranking functions, we have to develop the approach in order to handle nested loop programs, although it works well for single loop programs.

7 Conclusions and Future Work

In this paper, we reduced the polynomial invariant generation of polynomial programs to solving semi-algebraic systems, and theoretically analyzed why our approach is more efficient and practical than that of [17] directly applying the technique of first-order quantifier elimination. Compared to the well-established approaches in this field, the invariants generated with our approach are more expressive.

How to further improve the efficiency of our approach is still a big challenge as well as our main future work, as the complexity is still single exponential

w.r.t. the number of program variables and parameters, and doubly exponential w.r.t. the number of parameters (at least). The high complexity restricts to scale up our approach yet. Moreover, it deserves to investigate how to combine our approach with other program verification techniques such as abstract interpretation and Floyd-Hoare-Dijkstra's inductive assertion method in order to resolve complicated verification problems. In addition, implementing our approach in a verification tool also makes so much senses in practice.

Acknowledgements

The work of this paper is a continuation of the one in [2], we are therefore so grateful to Prof. Chaochen Zhou for his contribution to the previous joint work as well as lots of fruitful discussions on this work and valuable comments on the draft of this paper. Naijun Zhan's work owes much to Prof. Chaochen Zhou, who taught him formal methods and many other things.

References

1. Besson, F., Jensen, T., Talpin, J.-P.: Polyhedral analysis of synchronous languages. In: Cortesi, A., Filé, G. (eds.) SAS 1999. LNCS, vol. 1694, pp. 51–69. Springer, Heidelberg (1999)
2. Chen, Y., Xia, B., Yang, L., Zhan, N., Zhou, C.: Discovering non-linear ranking functions by solving semi-algebraic systems. In: ICTAC 2007. LNCS series (2007) (Invisted paper)
3. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Brakhage, H. (ed.) Automata Theory and Formal Languages. LNCS, vol. 33, pp. 134–183. Springer, Heidelberg (1975)
4. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. J. of Symbolic Computation 12, 299–328 (1991)
5. colón, M., Sankaranarayanan, S., Sipma, H.B.: Linear invariant generation using non-linear constraint solving. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 420–432. Springer, Heidelberg (2003)
6. Cousot, P.: Proving program invariance and termination by parametric abstraction, Langrangian Relaxation and semidefinite programming. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 1–24. Springer, Heidelberg (2005)
7. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among the variables of a program. In: ACM POPL'78, pp. 84–97 (1978)
8. Davenport, J.H., Heintz, J.: Real Elimination is Doubly Exponential. J. of Symbolic Computation 5, 29–37 (1988)
9. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall, Englewood Cliffs (1976)
10. Dolzman, A., Sturm, T.: REDLOG: Computer algebra meets computer logic. ACM SIGSAM Bulletin 31(2), 2–9
11. Floyd, R.W.: Assigning meanings to programs. Proc. Symphosia in Applied Mathematics 19, 19–37 (1967)
12. Gallo, G., Mishra, B.: Efficient Algorithms and Bounds for Wu-Ritt Characteristic Sets. In: Mora, T., Traverso, C. (eds.) Effective methods in algebraic geometry, Birkhäuser, Bosten. Progress in Mathematics, pp. 119–142 (1994)

13. German, S., Wegbreit, B.: A synthesizer of inductive assertions. IEEE Transactions on Software Engineering 1(1), 68–75 (1975)
14. Hoare, C.A.R.: An axiomatic basis for computer programming. Comm. ACM 12(10), 576–580 (1969)
15. Karr, M.: Affine relationships among variables of a program. Acta Informatica 6, 133–151 (1976)
16. Katz, S., Manna, Z.: Logical analysis of programms. Communications of the ACM 19(4), 188–206 (1976)
17. Kapur, D.: Automatically generating loop invariants using quantifier llimination. In: Proc. IMACS Intl. Conf. on Applications of Computer Algebra (ACA'04), Beaumont, Texas (July 2004)
18. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer, Heidelberg (1995)
19. Múller-Olm, M., Seidl, H.: Polynomial constants are decidable. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 4–19. Springer, Heidelberg (2002)
20. Múller-Olm, M., Seidl, H.: Precise interprocedural analysis through linear algebra. In: ACM SIGPLAN Principles of Programming Languages, POPL 2004, pp. 330–341 (2004)
21. Rodriguez-Carbonell, E., Kapur, D.: An abstract interpretation approach for automatic generation of polynomial invariants. In: Giacobazzi, R. (ed.) SAS 2004. LNCS, vol. 3148, pp. 280–295. Springer, Heidelberg (2004)
22. Rodriguez-Carbonell, E., Kapur, D.: Automatic generation of polynomial loop invariants: algebraic foundations. In: Proc. Intl. Symp on Symbolic and Algebraic Computation (ISSAC'04) (July 2004)
23. Rodriguez-Carbonell, E., Kapur, D.: Generating all polynomial invariants in simple loops. Journal of Symbolic Computation 42, 443–476 (2007)
24. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Non-linear loop invariant generation using Gröbner bases. In: ACM POPL'04, pp. 318–329 (2004)
25. Wegbreit, B.: The synthesis of loop predicates. Communications of the ACM 17(2), 102–112 (1974)
26. Wu, W.-T.: Basic principles of mechanical theorem proving in elementary geometries. J. Syst. Sci. Math. 4, 207–235 (1984)
27. Xia, B., Xiao, R., Yang, L.: Solving parametric semi-algebraic systems. In: Pae, S.-l, Park, H. (eds.) Proc. the 7th Asian Symposium on Computer Mathematics (ASCM 2005), Seoul, December 8-10, pp. 153–156 (2005)
28. Xia, B., Yang, L.: An algorithm for isolating the real solutions of semi-algebraic systems. J. Symbolic Computation 34, 461–477 (2002)
29. Xia, B., Zhang, T.: Real Solution Isolation Using Interval Arithmetic. Comput. Math. Appl. 52, 853–860 (2006)
30. Yang, L.: Recent advances on determining the number of real roots of parametric polynomials. J. Symbolic Computation 28, 225–242 (1999)
31. Yang, L., Hou, X., Xia, B.: A complete algorithm for automated discovering of a class of inequality-type theorems. Sci. in China (Ser. F) 44, 33–49 (2001)
32. Yang, L., Hou, X., Zeng, Z.: A complete discrimination system for polynomials. Science in China (Ser. E) 39, 628–646 (1996)
33. Yang, L., Xia, B.: Automated Deduction in Real Geometry. In: Chen, F., Wang, D. (eds.) Geometric Computation, pp. 248–298. World Scientific, Singapore (2004)
34. Yang, L., Xia, B.: Real solution classifications of a class of parametric semi-algebraic systems. In: Proc. of Int'l Conf. on Algorithmic Algebra and Logic, pp. 281–289 (2005)