Program Verification by Reduction to Semi-Algebraic Systems Solving [⋆]

Bican Xia[1], Lu Yang[2], and Naijun Zhan[3] [⋆⋆]

[1] LMAM & School of Mathematical Sciences, Peking University
[2] Shanghai Key Lab. of Trustworthy Computing, East China Normal University
[3] Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences

Abstract. The discovery of invariants and ranking functions plays a central role in program verification. In our previous work, we investigated invariant generation and non-linear ranking function discovering of polynomial programs by reduction to semi-algebraic systems solving. In this paper we will first summarize our results on the two topics and then show how to generalize the approach to discovering more expressive invariants and ranking functions, and applying to more general programs.

keywords Program Verification, Ranking Functions, Invariants, Polynomial Programs, Semi-Algebraic Systems, Quantifier Elimination

1 Introduction

The discovery of invariants and ranking functions plays a central role in program verification, and is therefore thought as the most challenging problem of program verification. In recent years, due to the advance of computer algebra, various approaches to non-linear invariant generation and termination analysis of polynomial programs have been established, based on computer algebra. These approaches have widely been applied to program verification and made tremendous success. However, almost each of these approaches has its limitations, e.g. some of them are limited to linear (affine) systems, some of them suffers from high complexity, some of them can only generate weak invariants or ranking functions and so on.

In order to overcome the weakness of the well-established approaches, following the line of [14], by exploiting our results on solving semi-algebraic systems (SASs), we proposed more practical and efficient approaches to polynomial invariant generation and ranking function discovering of polynomial programs respectively in [5] and [4]. In this paper, we will first summarize the results reported in [4, 5] and correct mistakes in Example 7 in [5]. Then we will extend

[⋆] This work is supported in part by NKBRPC-2002cb312200, NKBRPC-2004CB318003, NSFC-60493200, NSFC-60721061, NSFC-60573007, NSFC-90718041,and NSFC-60736017 and NKBRPC-2005CB321902.

[⋆⋆] The corresponding author: South Fourth Street, No. 4, Zhong Guan Cun, Beijing, 100080, P.R. China, `znj@ios.ac.cn` the approach such that it can not only be applicable to more general classes of programs and show how to synthesize more expressive invariants and ranking functions. We will investigate to extend the approach to general multivariate polynomial systems and even fractional polynomial systems. We also study to extend the approach to synthesizing invariants that can be represented by a general polynomial formula, even a fractional polynomial formula, and ranking functions that could be either a polynomial or a fractional polynomial.

1.1 Related work

Up to now, most of well-established invariant generation methods either based on abstract interpretation [10, 2, 18, 9], or on quantifier elimination [7, 14], or on polynomial algebra [15, 16, 19–21].

The basic idea of the approaches based on abstract interpretation is to perform approximate symbolic execution of a program until an assertion is reached that remain unchanged by further executions of the program. However, in order to guarantee termination, the method introduces imprecision by the 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 [10, 2].

In contrast, [15, 16, 19–21] exploited the theory of polynomial algebra to discover invariants of polynomial programs. The technique of linear algebra to generate polynomial equations of bounded degree as invariants of programs with affine assignments was applied in [15]. In [19, 20], it was first proved that the set of polynomials serving as loop invariants has the algebraic structure of an ideal, then was proposed an algorithm to obtain a finite base of the ideal by using fixpoint computation. Finally an algorithm by using Gröbner bases and the elimination theory was given. 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 is guaranteed to generate it. A similar approach to finding polynomial equation invariants whose form is priori determined (called templates) by using an extended Gröbner basis algorithm over templates was presented in [21].

Compared with polynomial algebra based approaches that can only generate invariants represented as polynomial equations, the approach from [7] can generate linear inequalities as invariants for linear programs, which based on *Farkas' Lemma* and non-linear constraint solving. In [14], a very general approach for automatic generation of more expressive invariants was proposed, which is based on 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 it was pointed out in [14] that this is impractical in reality because of the high complexity of quantifier elimination, which is at least double exponential [12].

Following the line of [14], a very general and efficient approach to ranking function discovery and invariance generation of linear and polynomial programs was presented in [9]. However, the approach of [9] 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.

Likewise, inspired by [14], by exploiting our results on solving semi-algebraic systems (SASs), we proposed more practical and efficient approaches to polynomial invariant generation and ranking function discovering of polynomial programs respectively in [5] and [4]. Comparing with other well-established invariant generation methods, the advantages of the approach of [5] include: Can generate more expressive invariants, which are represented as a semi-algebraic systems consisting of polynomial equations, inequations and inequalities; Has lower complexity, compared with the methods directly based on Gröbner Base or first-order quantifier elimination. The complexity of the approach in [5] is singly exponential in the number of program variables plus doubly exponential in the number of parameters approximately, while the latter with double exponential in the number of program variables and parameters; Still is complete, compared with the approach of [9] in the sense that whenever there exist invariants of the predefined form, our approach can indeed synthesize them.

On the other hand, compared to other well-established termination analysis approaches, the advantages of [4] include: Firstly, it can be applied to non-linear programs and discover non-linear ranking functions, whereas most of other well-established can only be applicable to linear programs and synthesize linear ranking functions, such as [11, 8, 17]; Secondly, the approach is complete compared with [9] in the sense that if there exist ranking functions of the predefined template, it can indeed discover them. But, our approach is just a sufficient method for termination analysis, not a sufficient and necessary one like [22, 1] which focuses on a special subclass of linear programs; Furthermore, the complexity of our approach is still very high comparing with the approaches of [9, 3].

1.2 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 order of the variables be $x_1 \prec x_2 \prec \cdots \prec x_n$. Then, the *leading variable* (or *main variable*) of a polynomial p is the variable with the greatest 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 $p = c_m x_k^m + \cdots + c_0$ 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 $p(x_1, \ldots, x_5) = x_2^5 + x_3^4 x_4^2 + (2x_2 + x_1)x_4^3$. The leading variable, term and coefficient of $p(x_1, \ldots, x_5)$ 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 $\rhd \in \{=, >, \geq, \neq\}$. A *polynomial formula* is a boolean combination of atomic polynomial formulae. We will denote by $PF(\mathcal{K}[x_1, \ldots, x_n])$ the set of polynomial formulae over $\mathcal{K}[x_1,...,x_n]$ and by $CPF(\mathcal{K}[x_1,\dots,x_n])$ the set of conjunctive polynomial formulae over $\mathcal{K}[x_1,...,x_n]$, which only contain logical connective $\wedge$, respectively.

An *atomic fractional polynomial formula* over $\mathcal{K}[x_1,...,x_n]$ is of the form $\frac{p(x_1,\dots,x_n)}{q(x_1,\dots,x_n)} \rhd 0$, where $p(x_1,\dots,x_n) \neq 0$ is relative prime to $q(x_1,\dots,x_n)$, both of them are in $\mathcal{K}[x_1,\dots,x_n]$, and $\rhd \in \{=,>,\geq,\neq\}$. A *fractional polynomial formula* is a boolean combination of atomic fractional polynomial formulae. We will denote by $FPF(\mathcal{K}[x_1,\dots,x_n])$ the set of fractional polynomial formulae over $\mathcal{K}[x_1,...,x_n]$.

It is easy to prove the following theorem that indicates $FPF(\mathcal{K}[x_1,\dots,x_n])$ is as expressive as $PF(\mathcal{K}[x_1,\dots,x_n])$.

Theorem 1. i) *$PF(\mathcal{K}[x_1,\dots,x_n]) \subseteq FPF(\mathcal{K}[x_1,\dots,x_n])$;*
ii) *For any $\phi \in FPF(\mathcal{K}[x_1,\dots,x_n])$, there exists $\phi' \in PF(\mathcal{K}[x_1,\dots,x_n])$ such that $\phi \Leftrightarrow \phi'$.*

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 brevity.

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

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

where $r > 0, l \geq k \geq 0, m \geq 0$ 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 $t \neq 0$, abbreviated as PSAS, otherwise* constant, *written as CSAS.*

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 CSAS 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 PSAS, 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 DISCOVERER

Theories on how to classify real roots of PSASs and isolate real roots of CSASs were developed in [26, 25, 24, 27]. The core of the theories is the generalized *Complete Discrimination System* (CDS). A computer algebra tool named DISCOVERER [23] has been developed in Maple to implement these theories. Comparing with other well-known computer algebra tools like REDLOG [13] and QEPCAD [6], DISCOVERER has two distinct features as follows.

Real Solution Classification of PSASs: For a PSAS T of the form (1) and an argument N 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 $b.. + \infty$, where b is a non-negative integer,

DISCOVERER provides the functions **tofind** and **Tofind**, which 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.

Real Solution Isolation of CSASs: For a CSAS T of the form (1) only with a finite number of real solutions, DISCOVERER can determine the number of distinct real solutions of T, say n, and 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 by calling **nearsolve** and **realzeros**, respectively.

3 Invariants and Ranking Functions

We use transition systems to represent programs.

Definition 2. *A 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, the transition relation ρ_τ is a first-order formula over $V \cup V'$, and θ_τ is a first-order formula over V, which is the guard of the transition. The location ℓ_0 is the initial location, and the initial condition Θ is a first-order formula over V.*

Only if θ_τ holds, the transition can take place. Here, we use V' (variables with prime) to denote the next-state variables.

If all formulae of a transition system are from CPF($\mathcal{K}[x_1, \ldots, x_n]$), the system is also called semi-algebraic transition system *(SATS). Similarly, a system is called* polynomial transition system *(PTS) (resp.* fractional polynomial transition system *(FPTS)), if all its formulae are in PF($\mathcal{K}[x_1, \ldots, x_n]$) (resp. FPF($\mathcal{K}[x_1, \ldots, x_n]$)).*

According to Theorem 1, it is easy to see that

Theorem 2. *For each fractional polynomial transition system, there is a polynomial transition system such that the two transition systems are equivalent in the sense that any property holds on one of them iff the property holds on the other either.*

A state is a valuation of the variables in V. The space of states is 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 transition systems can be explained through state transitions as usual.

We denote the transition $\tau = (l_1, l_2, \rho_\tau, \theta_\tau)$ by $l_1 \overset{\rho_\tau, \theta_\tau}{\rightarrow} l_2$, or simply by $l_1 \overset{\tau}{\rightarrow} l_2$. A sequence of transitions $l_{11} \overset{\tau_1}{\rightarrow} l_{12}, \ldots, l_{n1} \overset{\tau_n}{\rightarrow} l_{n2}$ is called *composable* if $l_{i2} = l_{(i+1)1}$ for $i = 1, \ldots, n-1$, and written as $l_{11} \overset{\tau_1}{\rightarrow} l_{12}(l_{21}) \overset{\tau_2}{\rightarrow} \cdots \overset{\tau_n}{\rightarrow} l_{n2}$. A composable sequence is a *transition circle* at l_{11}, if $l_{11} = l_{n2}$. For any composable sequence $l_0 \overset{\tau_1}{\rightarrow} l_1 \overset{\tau_2}{\rightarrow} \cdots \overset{\tau_n}{\rightarrow} l_n$, it is easy to show that there is a transition of the form $l_0 \overset{\tau_1;\tau_2;\cdots;\tau_n}{\rightarrow} l_n$ such that the composable sequence is equivalent to the transition, where $\tau_1; \tau_2 \cdots; \tau_n$, $\rho_{\tau_1;\tau_2;\cdots;\tau_n}$ and $\theta_{\tau_1;\tau_2;\cdots;\tau_n}$ are the compositions of $\tau_1, \tau_2, \ldots, \tau_n$, $\rho_{\tau_1}, \ldots, \rho_{\tau_n}$ and $\theta_{\tau_1}, \ldots, \theta_{\tau_n}$, respectively. The composition of transition relations is defined in the standard way, for example, $x' = x^4 + 3; x' = x^2 + 2$ is $x' = (x^4 + 3)^2 + 2$; while the composition of transition guards have to be given as a conjunction of the guards, each of which takes into account the past state transitions. In the above example, if the guard of the first transition is $x + 7 = x^5$ and that of the second is $x^4 = x + 3$, then guard of the composition is $x + 7 = x^5 \wedge (x^4 + 3)^4 = (x^4 + 3) + 3$.

3.1 Invariants

Informally, an *invariant* of a program at a location is an assertion that holds on 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, *initiation* and *consecution. Initiation* means that the image of the mapping at the initial location holds on the loop entry; while *consecution* 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 *initiation* and *consecution* mean that the invariant holds on the loop entry, and is preserved by any transition circle at the entry point.

Definition 3 (Invariant at a Location). *Let $P = \langle V, L, T, l_0, \Theta \rangle$ be a transition system. An invariant at a location $l \in L$ is a first-order formula ϕ over V such that ϕ holds on all states that can be reached at location l.*

Definition 4 (Invariant of a Program). *An* assertion map *for a transition system $P = \langle V, L, T, l_0, \Theta \rangle$ is a map associating each location of P with a first-order formula. An assertion map η of P is said to be* inductive *iff the following conditions hold:*

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

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

It is well-known that if η is an inductive mapping of P, then $\eta(l)$ is an invariant of P at l. Therefore, an inductive mapping of P forms an invariant of P.

3.2 Ranking Functions

Definition 5 (Ranking Function). *Assume $P = \langle V, L, T, l_0, \Theta \rangle$ is a transition system. A ranking function is a function $\gamma : Val(V) \rightarrow \mathbb{R}^+$ such that the following conditions are satisfied:*

Initiation: $\Theta(V_0) \models \gamma(V_0) \geq 0$.
Decreasing: *There exists a constant $C \in \mathbb{R}^+$ such that $C > 0$ and for any transition circle $l_0 \xrightarrow{\tau_1} l_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_{n-1}} l_{n-1} \xrightarrow{\tau_n} l_0$ at l_0,*

$$\rho_{\tau_1;\tau_2;\cdots;\tau_n}(V, V') \wedge \theta_{\tau_1;\tau_2;\cdots;\tau_n}(V) \models \gamma(V) - \gamma(V') \geq C \wedge \gamma(V') \geq 0.$$

Condition 1 says that the ranges of all the initial states satisfying the initial condition under the ranking function is nonnegative; Condition 2 expresses the fact that the value of the ranking function decreases by at least C as the program moves back to the initial location along any transition circle, and is still nonnegative.

In Definition 5, if γ is a polynomial, then it is called a *polynomial ranking function.*

Remark 1.
- According to Definition 5, for any transition system, if a ranking function exists, then the system will not go through l_0 infinitely often.
- Ranking functions can be seen as loop invariants at the entry point.

In the subsequent two sections, we will summarize the results of [5, 4], where all formulae are in $CPF(\mathcal{K}[x_1, \ldots, x_n])$ and ranking functions are polynomial.

4 Generating Polynomial Invariants

Given an SATS S, the procedure of generating polynomial invariants with the approach of [5] includes the following 4 steps:

1. **Predefine Parametric Invariants** Predefine a template of invariants at each of the underlining locations, which is a PSAS. All of these predefined PSASs form a parametric invariant of the program.
2. **Derive PSASs from Initial Condition and Then Solve** According to Definition 4, 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 common real solutions. This implies that for each atomic polynomial formula ϕ in $\eta(l_0)$, $\Theta \wedge \neg\phi$ has no real solutions. Note that $\eta(l_0)$ is the conjunction of a set of atomic polynomial formulae and therefore $\Theta \wedge \neg\phi$ is a PSAS according to the definition. Thus, applying the tool DISCOVERER to the resulting PSAS $\Theta \wedge \neg\phi$, we get a necessary and sufficient condition such that the derived PSAS has no real solutions. 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 universally quantifying these variables (we usually add a scope to each of these universally quantified variables according to the

program) 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 use the conjunction of all the resulting conditions.

3. **Derive PSASs from Consecutive Condition and Then Solve** From Definition 4, 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 solutions, which implies that for each atomic polynomial formula ϕ in $\eta(l_j)$,
$$\eta(l_i) \wedge \rho_\tau \wedge \theta_\tau \wedge \neg\phi \qquad (2)$$
has no real solution. It is clear that (2) is a PSAS. By applying the tool DISCOVERER, we obtain a necessary and sufficient condition on the parameters for (2) to have no real solution. Similarly to Step 2, we may need to use quantifier elimination in order to get a necessary and sufficient condition only on the presumed parameters.
4. **Generate Invariants** 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.

Note that the above procedure is *complete* in the sense that for any given predefined parametric invariant, the procedure can always produce the corresponding concrete invariant, if it exists. Therefore, we can also 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, because we can assume parametric invariants in program variables of different degrees, and repeatedly apply the above procedure until we obtain a polynomial invariant.

Remark 2. In Steps 2 and 3, we can also apply DISCOVERER for first-order quantifier elimination in a special manner. We will illustrate this point by the following example.

4.1 Example

In this subsection, we will illustrate the above procedure by revising Example 7 from [5] which contains some mistakes in the resulting conditions because of incorrect use of the tools DISCOVERER and QEPCAD.

Example 1. The code of the program is on Fig.4 (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.4

The corresponding SATS is on Fig.4 (b).

Firstly, we predefine a parametric invariant at l_0 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, $\eta(l_0) = (3) \wedge (4)$.

Secondly, according to *Initiation* of Definition 4, $\Theta \models \eta(l_0)$ is equivalent to neither of the following two PSASs having real solutions.

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

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

By calling **tofind**$(([x, y], [\,], [\,], [eq(x, y)], [x, y], [a_1, a_2, a_3, a_4], 0)$ we get that (5) has no real solutions iff *true*. While calling **tofind**$([x, y], [-ineq(x, y)], [\,], [\,], [x, y], [b_1, b_2, b_3, b_4], 0)$ we get that (6) has no real solutions iff $b_4 > 0$.

Thirdly, consider Consecution w.r.t. the transition τ. We have

$$eq(x, y) = 0 \wedge x' - x - y^2 = 0 \wedge y' - y - 1 = 0 \models eq(x', y') = 0 \wedge ineq(x', y') > 0. \tag{7}$$

This means that the following two PSASs both have no real solutions.

$$eq(x, y) = 0 \wedge x' - x - y^2 = 0 \wedge y' - y - 1 = 0 \wedge x \geq 0 \wedge y \geq 0 \wedge eq(x', y') \neq 0 \tag{8}$$

$$ineq(x, y) > 0 \wedge x' - x - y^2 = 0 \wedge y' - y - 1 = 0 \wedge x \geq 0 \wedge y \geq 0 \wedge ineq(x', y') \leq 0 \tag{9}$$

By calling **tofind**$([x' - x - y^2, y' - y - 1, eq(x, y)], [x, y], [\,], [eq(x', y')\ ], [x', y', x], [y, a_1, a_2, a_3, a_4], 0)$, we obtain that (8) has no real solutions if and only if

$$a_3 y^2 + 3a_1 y^2 + 2ya_2 + 3a_1 y - a_4 + a_2 + a_1 = 0 \vee \tag{10}$$

$$a_3 = 0.[4] \tag{11}$$

Further by Basic Algebraic Theorem, (10) holds for all y iff

$$-a_4 + a_2 + a_1 = 0 \wedge 3a_1 + 2a_2 = 0 \wedge a_3 + 3a_1 = 0, \tag{12}$$

while (11) leads to a trivial result.

For (9), by calling **tofind**$([x' - x - y^2, y' - y - 1], [-ineq(x', y'), x, y], [ineq(x, y)], [\,],$

[4] This resulting condition on a_1, a_2, a_3, a_4 from Consecution is different from the one given in [5], as there happened a mistake in calling **tofind** in [5]. But the final condition in [5] is correct, same as here.

$[x', y'], [x, y, b_1, b_2, b_3, b_4], 0)$, we obtain that (9) has no real solutions iff

$$b_4 + b_3 + b_2 + 2b_2 y + b_3 y + b_2 y^2 + b_1 x + b_1 y^2 > 0. \tag{13}$$

Then, we have to perform quantifier elimination on (13) under the premise that $x \geq 0, y \geq 0, ineq(x, y) > 0$. Here, we use DISCOVERER in a complicated way [5], and get the following sufficient and necessary condition[6]

$$\begin{gathered} b_1 > 0 \wedge b_4 > 0 \wedge b_2 + b_3 + b_4 > 0 \wedge ((b_2 \geq 0 \wedge b_2 + b_3 \geq 0) \vee \\ (b_1 + b_2 \geq 0 \wedge 2b_2 + b_3 \geq 0) \vee d_1 \leq 0 \vee d_2 < 0 \vee (f_1 \leq 0 \wedge f_2 \geq 0)) \end{gathered} \tag{14}$$

where

$$\begin{aligned} d_1 &= -4b_1 b_3 - 4b_1 b_2 + 4b_2^2 \\ d_2 &= -4b_1 b_2 - 4b_1 b_3 - 4b_4 b_1 + b_3^2 - 4b_4 b_2 \\ f_1 &= 2b_4 b_1^2 + 2b_4 b_1 b_2 - 2b_2^2 b_1 - 4b_1 b_2 b_3 - b_1 b_3^2 + 2b_2^3 \\ f_2 &= b_2^4 - 2b_2^2 b_1 b_4 + b_4^2 b_1^2 - 4b_4 b_1 b_3 b_2 - b_2^2 b_3^2 + 4b_2^3 b_4 + b_1 b_3^3 + b_1 b_2 b_3^2 \end{aligned}$$

Remark 3. We also applied QEPCAD to the above formula and obtained a different formula, though equivalent formula.

It is easy to see that the invariant given in [5] is still an invariant of the program with the predefined template, i.e.

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

is an invariant of P, where

$$(a_1, a_2, a_3, a_4) = (-2, 3, 6, 1), (b_1, b_2, b_3, b_4) = (1, -1, 2, 1).$$

Furthermore,

$$\begin{cases} -2y^3 + 3y^2 + 6x - y = 0, \\ \frac{4}{3}x - y^2 + \frac{1}{4}y + 3 > 0 \end{cases}$$

is another invariant of P , where

$$(a_1, a_2, a_3, a_4) = (-2, 3, 6, 1), (b_1, b_2, b_3, b_4) = (\frac{4}{3}, -1, \frac{1}{4}, 3).$$

However, the latter invariant does not satisfy the formula (17) in [5].

[5] The procedure is so involved, as we need to apply the tool multiple times, so we here omit the detailed discussion.

[6] The formula (17) in [5] is not correct because of a mistake when using QEPCAD.

5 Discovering Non-linear Ranking Functions

As we explained in Remark 1, a ranking function can be represented as a special loop invariant of a loop at the entry point. Therefore, the above procedure still works for non-linear ranking function discovering subject to appropriate modifications. Roughly speaking, The approach to discovering non-linear ranking functions in [4] consists of the following 4 steps:

Step 1–Predefine a Ranking Function Template Predetermine a template of ranking functions.

Step 2– Encode Initial Condition According to the initial condition of ranking function, we have $\Theta \models \gamma \geq 0$ which means that each real solution of Θ must satisfy $\gamma \geq 0$. In other words, $\Theta \wedge \gamma < 0$ has no real solution. It is easy to see that $\Theta \wedge \gamma < 0$ is a PSAS according to Definition 1. Therefore, by applying DISCOVERER, we get a necessary and sufficient condition for the derived PSAS to have no real solutions. The condition may contain the occurrences of some program variables. In this case, the condition should hold for any instantiations of the variables. Thus, by introducing universal quantifications of these variables (we usually add a scope to each of the variables according to different situations) and then applying QEPCAD or DISCOVERER, we can get a necessary and sufficient condition in terms of the parameters only.

Step 3–Encode Decreasing Condition From Definition 5, there exists a positive constant C such that for any transition circle $l_0 \xrightarrow{\tau_1} l_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_n} l_0$,

$$\rho_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \theta_{\tau_1;\tau_2;\cdots;\tau_n} \models \gamma(V) - \gamma(V') \geq C \wedge \gamma(V') \geq 0, \tag{15}$$

equivalent to

$$\rho_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \theta_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \gamma(V') < 0, \tag{16}$$

$$\rho_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \theta_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \gamma(V) - \gamma(V') < C \tag{17}$$

both have no real solutions. Obviously, (16) and (17) are PSASs according to Definition 1. Thus, by applying DISCOVERER, we obtain some conditions on the parameters. Subsequently, similarly to Step 2, we may need to use QEPCAD or DISCOVERER to simplify the resulting condition in order to get a necessary and sufficient condition in terms of the parameters only.

Step 4–Solve Final Constraints 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 the ranking function template. Then, by utilizing DISCOVERER or QEPCAD, prove if or not the condition is satisfied and produce the instantiations of these parameters such that the condition holds. Thus, we can get a ranking function of the predetermined form by replacing the parameters with the instantiations, respectively.

Remark 4. Note that the above procedure is *complete* in the sense that for any given template of ranking function, the procedure can always synthesize a ranking function of the give template, if there indeed exist such ranking functions.

5.1 Discussions: Generating Invariants vs Discovering Ranking Functions

We have shown how to reduce the discovery of invariants and ranking functions to directly solving SASs by exploiting the inductive property of invariants and ranking functions. Although invariants and ranking functions both have such a property, the former is inductive w.r.t. a small step, i.e. each of single transitions of the given loop in contrast that the latter is inductive w.r.t. a big step, that is each of transition circle at the initial location of the loop. The difference entails that the approach from [5] 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 further develop the approach of [4] in order to handle nested loop programs, although it works well for single loop programs.

6 Complexity Analysis

Assume given an SATS $P = \langle V, L, T, l_0, \Theta \rangle$, we obtain k distinct PSASs in order to generate its polynomial invariants or ranking functions with the approach. 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 CAD (*cylindrical algebraic decomposition*) based quantifier elimination on S has complexity $\mathcal{O}((2d)^{2^{2n+8}}(s+m)^{2^{n+6}})$ according to the results of [12], 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 invariants and ranking functions of a program as advocated by Kapur [14].

In contrast, the cost of our approach includes two parts: one is for applying real solution classification to generate condition on the parameters possibly still containing 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. According to the complexity analysis in [5], the cost for the first part is singly exponential in n and doubly exponential in t, where t stands for the dimension of the ideal generated by the s polynomial equations. The cost for the second part is doubly exponential in t. So, compared to directly applying quantifier elimination, our approach can dramatically reduce the complexity, in particular when t is much less than n.

7 Beyond Semi-Algebraic Transition Systems

In this section, we will discuss how to generalize the above approach to more general programs beyond SATSs.

Polynomial Transition Systems A PTS can be transformed into an equivalent SATS by adding additional transitions. The basic idea is as follows: First, given a PTS P, we rewrite all guards and transitions relations in disjunctive normal form. Let P' be the resulting PTS; Second, if there is a transition of the form $\tau = \langle i, j, \rho_\tau, \theta'_\tau \vee \theta''_\tau \rangle$, then we replace τ by $\tau_1 = \langle i, j, \rho_\tau, \theta'_\tau \rangle$ and $\tau_2 = \langle i, j, \rho_\tau, \theta''_\tau \rangle$; if there is a transition of the form $\tau = \langle i, j, \rho'_\tau \vee \rho''_\tau, \theta_\tau \rangle$, then we replace τ by $\tau_1 = \langle i, j, \rho'_\tau, \theta_\tau \rangle$ and $\tau_2 = \langle i, j, \rho''_\tau, \theta_\tau \rangle$; Repeat the second step until all guards and transition relations are conjunctive polynomial formulae. Finally, we obtain an SATS that is equivalent to P.

The following theorem guarantees that we can reduce the problem of discovery of invariants and ranking functions of a PTS to that of the resulting SATS.

Theorem 3. *Let P be a PTS, and P' be the resulting SATS from the above procedure. Then, P and P' have the same invariants and ranking functions.*

Fractional Polynomial Transition Systems According to Theorem 2 and the above discussion, it is easy to see that the problems of invariant generation and ranking function discovering for FPTSs can be reduced to those of SATSs too.

8 More Expressive Invariants and Ranking Functions

In this section, we will discuss how to extend the approach to synthesizing more expressive invariants and ranking functions. According to the results of the above section, for simplicity, here we only need to consider to extend the approach to synthesize more expressive invariants and ranking functions of SATSs.

General Polynomial Formula as Invariant Given an SATS, we will extend the approach presented in Section 4 in the following way: In first step, we allow to predefine a template of invariant ϕ, which is a parametric polynomial formula rather than a PSAS. Then, we rewrite the parametric polynomial formula into a conjunctive normal form $\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$, where each ϕ_i is a disjunction of some atomic polynomial formulae. In the second step, according to Initiation, we have $\Theta_0 \models \bigwedge_{i=1}^n \phi_i$. This means that $\Theta_0 \wedge \bigvee_{i=1}^n \neg\phi_i$ has no real solutions. This entails that for $i = 1, \cdots, n$, $\Theta_0 \wedge \neg\phi_i$ has no real solutions. It is easy to see that $\Theta_0 \wedge \neg\phi_i$ is a PSAS, therefore, Initiation case is reduced to solving SASs. Applying the technique used in Section 4, we can obtain a condition on the parameters only. In the third step, similarly to the above, we can show that Consecution can also be reduced to solving SASs either, and therefore, we can get another condition on the parameters. Finally, similarly to Section 4, we can instantiate the parameters according to the resulting condition on them and generate invariants with the predefined template.

Fractional Polynomial Formula as Invariant For this case, in the first step, we predefine a template of invariant that is a parametric fractional polynomial formula. According to Theorem 1, the parametric fractional polynomial formula is equivalent to a parametric polynomial formula. So, the rest steps are reduced to those of the above case.

Fractional Polynomial as Ranking Function By Theorem 1, similarly to the previous discussion, it is quite easy to extend the approach of [4] to synthesize a ranking function represented by a fractional polynomial.

9 Conclusions

In this paper, we first summarized our previous work on synthesizing polynomial invariants and ranking functions reported in [4, 5] by reduction to solving SASs, and redid Example 7 from [5] by correcting some mistakes. Then, we investigated the issue to generalize the approach in two directions: one is applicable to more general programs, beyond SATSs, to PTSs, even to FPTSs; the other is to synthesize more expressive invariants and ranking functions.

How to further improve the efficiency of the approach is still a big challenge as well as our main future work, since the complexity is still single exponential w.r.t. the number of program variables, and doubly exponential w.r.t. the number of parameters (at least). It is worth investigating how to further extend the approach to handle more general programs with more complicated data. The potential solution could be to integrate different decision procedures. Here, we only focus on Tarski's Algebra, so we can only deal with real variables.

Acknowledgements

We are so grateful to Prof. Chaochen Zhou and Yinhua Chen for their contributions to the previous joint work. We also thank Prof. Chaochen Zhou for many fruitful discussions with him on this work and his valuable comments on the draft of this paper. We thank Dr. Dimitar P. Guelev for his proof-reading and comments to improve the presentation of this paper so much.

References

1. M. Braverman. Termination of integer linear programs. In Proc. CAV'06, LNCS 4144, pp. 372-385. 2006.
2. F. Besson, T. Jensen, and J.-P. Talpin. Polyhedral analysis of synchronous languages. In SAS'99, LNCS 1694, pp. 51-69, Springer-Verlag, 1999.
3. A. Bradley, Z. Manna and H. Sipma. Terminaition of polynomial programs. In Proc. of VMCAI'05, LNCS
4. Y. Chen, B. Xia, L. Yang, N. Zhan and C. Zhou. Discovering non-linear ranking functions by solving semi-algebraic systems. ICTAC'07, LNCS 4711, pp.34-49.
5. Y. Chen, B. Xia, L. Yang, and N. Zhan. Generating polynomial invariants with DISCOVERER and QEPCAD. In Proc. of Formal Methods and Hybrid Real-Time Systems'07 (the Festschrift Symposium for Dines Bjorner and Zhou Chaochen), LNCS 4700, pp.67-82.

6. G. E., Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. of Symbolic Computation*, 12:299–328, 1991.
7. M. Colón, S. Sankaranarayanan and H.B. Sipma. Linear invariant generation using non-linear constraint solving. In CAV'03, LNCS 2725, pp. 420–432, 2003.
8. M. Colón and H.B. Sipma. Synthesis of linear ranking functions. In TACAS'01, LNCS 2031, pp. 67–81, 2001.
9. P. Cousot. Proving program invariance and termination by parametric abstraction, Langrangian Relaxation and semidefinite programming. In VMCAI'05, LNCS 3385, pp. 1-24. 2005.
10. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among the variables of a program. In ACM POPL'78, pp. 84-97, 1978.
11. D. Dams, R. Gerth, and O. Grumberg. A heuristic for the automatic generation of ranking functions. In *Workshop on Advances in Verification (WAVe'00)*, pp. 1-8, 2000.
12. J. H., Davenport and J. Heintz. Real Elimination is Doubly Exponential. *J. of Symbolic Computation,* 5:29–37, 1988.
13. A. Dolzman and T. Sturm. REDLOG: Computer algebra meets computer logic. *ACM SIGSAM Bulletin,* 31(2):2–9.
14. D. Kapur. Automatically generating loop invariants using quantifier llimination. In Proc. IMACS Intl. Conf. on Applications of Computer Algebra (ACA'04), Beaumont, Texas, July 2004.
15. M. Müller-Olm and H. Seidl. Polynomial constants are decidable. 9th Static Analysis Symposium (SAS'02), LNCS 2477, pp. 4-19. 2002.
16. M. Müller-Olm and H. Seidl. Precise interprocedural analysis through linear algebra. ACM SIGPLAN Principles of Programming Languages, POPL'04, pp. 330-341. 2004.
17. A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In VMCAI'04, LNCS 2937, pp. 239–251, 2004.
18. E. Rodriguez-Carbonell and D. Kapur. An abstract interpretation approach for automatic generation of polynomial invariants. In Proc. Static Analysis Symposium (SAS'04), LNCS 3148, pp. 280-295. August 2004.
19. E. Rodriguez-Carbonell and D. Kapur. Automatic generation of polynomial loop invariants: algebraic foundations. In. Proc. Intl. Symp on Symbolic and Algebraic Computation (ISSAC'04). July 2004.
20. E. Rodriguez-Carbonell and D. Kapur. Generating all polynomial invariants in simple loops. *Journal of Symbolic Computation*, 42:443-476. 2007.
21. S. Sankaranarayanan, H.B. Sipma, and Z. Manna. Non-linear loop invariant generation using Gröbner bases. In ACM POPL'04, pp. 318–329, 2004.
22. A. Tiwari. Termination of linear programs. In CAV'04, LNCS 3114, pp. 70–82, 2004.
23. B. Xia. DISCOVERER: A tool for solving semi-algebraic systems, *Software Demo at ISSAC 2007*, Waterloo, July 30, 2007. Also: *ACM SIGSAM Bulletin*, 41(3):102–103, Sept., 2007.
24. B. Xia and L. Yang. An algorithm for isolating the real solutions of semi-algebraic systems. *J. Symbolic Computation,* 34:461–477, 2002.
25. L. Yang. Recent advances on determining the number of real roots of parametric polynomials. *J. Symbolic Computation*, 28:225–242, 1999.
26. L. Yang, X. Hou and Z. Zeng. A complete discrimination system for polynomials. *Science in China (Ser. E)*, 39:628–646, 1996.

27. L. Yang and B. Xia. 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.
28. L. Yang, N. Zhan, B. Xia and C. Zhou. Program verification by using DISCOVERER. In the Proc. VSTTE'05, LNCS 4171, pp.575-586.