Front. Comput. Sci. China, 2008, 2(3)
DOI 10.1007/s11467-008-0000-0

REVIEW ARTICLE

Recent Advances in Program Verification Through Computer Algebra

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

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

Abstract In this paper, we summarize the results on program verification through semi-algebraic systems solving that we have obtained, including automatic discovery of invariants and ranking functions, symbolic decision procedure for the termination of a class of linear loops, termination analysis of nonlinear systems, and so on.

Keywords program verification, computer algebra, semi-algebraic systems solving, embedded systems, invariants, ranking functions, termination

1 Motivation

In this section, we will explain the motivation of this work and introduce the related work.

1.1 The design of trustworthy software is a grand challenge

As the modern society becomes more and more computerized, it is not taken granted to claim that we could not live without computer, as computer has been applied to every area in our life from the control panel of washing machine to the control system of rocket and so on. In particular, many of these applications are so-called *safety critical systems* [24], any fault happening in them may result in catastrophic consequences. For instances, Ariane 5's maiden flight (Ariane 5 Flight 501) on 4 June 1996 failed with the rocket self-destructing in 37 seconds after the launch because of a malfunction in the control software [35]; The Patriot missile [51] missed the goal resulting in nearly 30 soldiers died in the first Gulf War because of rounding error; The Mars Exploration Rover "Spirit" suffered a debilitating anomaly that prevented communication with Earth for several anxious days [46] because of its file system, etc. How to design these software systems and guarantee them trustworthy, that is, these systems exactly behave as what are expected, is a grand challenge in computer science [52].

Formal methods are based on rigor mathematics and thought as an effective way to attack the challenge, and have been made great success in practice. So far, three formal techniques for designing and verifying trustworthy software have been well-established, and widely and successfully applied in practice, i.e. model-checking [11,45], theorem proving [41–43] and abstraction interpretation [18,19].

1.2 Classical program theories are not enough for the new challenge

Classical program theories are mainly based on discrete mathematics, and have been well developed to handle discrete variables. However, in recent years, computers have been applied to numerous systems with continuous behaviour, in particular so-called *hybrid systems* which are the combination of continuous and discrete behaviours. Most of these systems are also embedded in physical devices (therefore also called *embedded systems*) and demand critical safety (thus also called *safety-critical systems*). Clearly, these systems cannot be well designed and verified using classical program theories. Thus, it is desirable as well as a challenge to look for new program theories that accommodate continuous mechanisms. The first attempt towards the challenge is due to Zhou et al by introducing *integral* into interval temporal logic [37] and establishing Duration Calculus [71]. Inspired by Dura-

tion Calculus, some other attempts have also been done, e.g. hybrid automata [1], TL$\int$ [36], TLA$^+$ [32], and hybrid statecharts [30], and so on, by introducing *differential or integral equation* to model continuous behaviour of systems.

Absolutely, the design of trustworthy software needs richer mathematical knowledge. In fact, recently more and more mathematical theories are found to be quite useful in program verification. For example, the theory of Gröbner Base is successfully applied to invariant generation of programs [47,48,50]; The first-order quantifier elimination techniques, mainly based on *cylindrical algebra decomposition* [12,13], have been used in reachability computation of linear hybrid systems [33,69] and program verification [29], and so on.

1.3 A challenge from verification of non-linear systems

Non-linear programs (systems) are a very important class of programs, in particular in scientific computing. However, most well-established work concentrates on linear programs (systems). For examples, most results on hybrid systems are on linear hybrid systems and little on nonlinear ones (of course lots of them can be approximated by linear systems [25]); Most termination analysis work focus on special classes of linear programs [44, 53]; And so on.

In general, the verification of non-linear systems is much more difficult, and needs richer and deeper mathematical knowledge, in particular continuous mathematics. How to verify nonlinear systems is still a challenging problem.

1.4 Synopsis

Our efforts commenced five years ago, when Lu Yang and Bican Xia, two experts on computer algebra, and Chaochen Zhou and Naijun Zhan, two experts on computer science were encouraged by the similar research in other countries and realised that it would be interesting to investigate if the theories and tools on computer algebra developed by Chinese mathematicians and computer scientists can successfully be applied to program verification. With the fund of Natural Science Foundation of China (NSFC) under the grant 60573007 "Application of Real Algebraic and Symbolic Computation to Formal Methods", we four persons formed an interdisciplinary team and started our attempt. The research was then further stressed by the key project "Analysis, Verification and Tools on Embedded Softwares based on Computer Algebra" under the grant 90718041 within the framework of "Basic Research on Trustworthy Software" funded by NSFC. At the same time, our team was strengthened as new participants joined.

In a word, what we have obtained up to now include:

- By exploiting our team members' algorithm on solving semi-algebraic systems (SASs), in particular on root classification of parametric SASs [65,67,68] and real root isolation of constant SASs [58,60], we have proposed a more practical and efficient approach to discovery of invariants and ranking functions of polynomial programs over real closed fields. Compared with other well-established work in this area, the advantages of our approach lie in: First, it can efficiently generate more expressive invariants; Second, our approach can be applied to nonlinear programs and discover nonlinear ranking functions, in contrast with that other well-known approaches can only be applied to linear programs and discover linear ranking functions; Finally, our approach is also complete in the sense that it can always tell you whether there exists a demanded polynomial invariant (or ranking function), and generate it if so.

  The outline of our approach is as follows: We first predefine a polynomial template of invariant (ranking function); Then we reduce invariant generation (ranking function discovering) to semi-algebraic system solving; After this we apply our theories and tools on solving SASs to produce some necessary and sufficient conditions; And finally utilize the technique of quantifier elimination to handle the derived conditions and obtain invariants (resp. ranking functions) with the predefined form.
- We Revisited the results reported in [53] and gave a symbolic decision procedure for termination of the class of linear loops considered in [53]. We showed that the complexity of our symbolic decision procedure is not higher than the numerical one in [53]. In particular, we invented a symbolic decision procedure with polynomial complexity for the special case, where the characteristic polynomial of the assignment matrix of a given loop is irreducible.
- We considered the termination problems of some classes of nonlinear programs. First, we showed that the termination problem of the following non-linear program $\tilde{\mathcal{P}}_1$ is decidable under a reasonable assumption, and conjectured it is undecidable in general.

$$\tilde{\mathcal{P}}_1: \quad \textbf{while} \quad (P(X) > b) \quad \{X := AX + C\},$$

  where $X = [x_1 \ \ldots \ x_N]^{\mathrm{T}}$ is the vector of program variables, $P(X) = [P_1(X) \ \ldots \ P_M(X)]^{\mathrm{T}} > b$ are polynomial constraints, each $P_i(X)(1 \leq i \leq M)$ is a polynomial in $\mathbb{Q}[X]$, and A is an $N \times N$ matrix over $\mathbb{Q}$ (the rational numbers). Second, we considered the termination problem of so-called *loop over intervals*, i.e.

$$\mathcal{P}_\Omega: \quad \textbf{while} \quad (x \in \Omega) \quad \{x := f(x)\}$$

  where x is the only program variable, Ω is an interval and f is a continuous function. In [70], Yao

established necessary and sufficient conditions for the termination problem of the program, and proved that the termination is decidable if f is a piecewise polynomial function.

- Some other results, for examples, in [63], Xu, Chen and Li investigated the reachability computation of non-linear hybrid systems; while in [64], Xu et al gave a method on how to compute reachable sets of rational eigenvalue linear hybrid systems; in [69], we proposed an off-line approach to generate sufficient termination conditions for linear loops. Also, in [69], we showed how to use DISCOVERER to improve the efficiency of the computation of reachable sets of linear hybrid systems; etc.

In this paper, we will briefly introduce the results listed in the first three items above.

1.5 Related work

Work on program verification can date back to the late sixties (or early seventies) of the 20th century when the so-called *Floyd-Hoare-Naur's inductive assertion method* [22,26,40] was invented, which was thought as the dominant approach on automatic program verification. The method is based on Hoare Logic [26], by using *pre-* and *post- conditions*, *loop invariants* and *termination analysis* through ranking functions, etc. Therefore, the discovery of loop invariants and ranking functions plays a central role in proving the correctness of programs and is also thought of as the most challenging part of the approach.

Since then, there have been lots of attempts to handle invariant generation of programs, e.g. [23,27,28,54], 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 [7,16,17,47], quantifier elimination [14,29] and polynomial algebra [38,39,48–50].

The basic idea behind the approaches based on abstract interpretation is to perform an 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 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 [7,17].

In contrast, approaches by exploiting the theory of polynomial algebra to discover invariants of polynomial programs were proposed in [38, 39, 48–50]. In [38], Mueller-Olm and Seidl applied the technique of linear algebra to generate polynomial equations of bounded degree as invariants of programs with affine assignments. In [48, 49], Rodrigez-Carbonell and Kapur first proved that the set of polynomials serving as loop invariants has the algebraic structure of 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. Sankaranarayanan et al in [50] presented a similar approach to finding polynomial equation invariants whose form is priori determined (called templates) by using an extended Gröbner basis algorithm.

Compared with polynomial algebra based approaches that can only generate invariants represented as polynomial equations, Colón et al in [14] proposed an approach to generate linear inequalities as invariants for linear programs, based on *Farkas' Lemma* and nonlinear constraint solving. In addition, Kapur in [29] 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 Kapur also pointed out in [29] that this is impractical in reality because of the high complexity of quantifier elimination, which is double exponential [21].

The classical method for establishing termination of a program is the use of well-founded domain together with so-called ranking function that maps the state space of the program to the domain. Termination is then concluded by demonstrating that each step as the program moves forwards decreases the measure assigned by the ranking function. As there can be no infinite descending chain of elements in a well-founded domain, any execution of the program must eventually terminate. Clearly, the existence of such a ranking function for any given program implies its termination. Recently, the synthesis of ranking functions draws increasing attention, and some heuristics concerning how to automatically generate linear ranking functions for linear programs have been proposed [15,20,44]. Dams et al in [20] proposed a heuristic strategy to synthesize a linear ranking function according to the syntax of a linear program. Since in many cases there does not exist obvious correlation between the syntax of a program and its ranking functions, this approach is very restrictive. Notably, Colón and Sipma in [15] utilized the theory of polyhedra to synthesize linear ranking function of linear programs, and Poldelsiki and Rynbalchenko in [44] first presented a complete method to find out linear ranking functions for a class of linear pro-

grams that have only single path without nested loop, in the sense that if there exists a linear ranking function for a program in the class, then the method can discover it.

Existence of ranking function is only a sufficient condition on the termination of a program. It is easy to construct programs that terminate, but without ranking functions. Furthermore, in [9] we found that even if a program has ranking functions, it may not have any linear ones. Besides, it is well-known that the termination of programs is undecidable in general, even for the class of linear programs [53] or a simple class of polynomial programs [8]. In contrast to the above approach, Tiwari [53] identified a useful class of linear programs and proved the decidability of the termination problem of the class over reals, while Braverman continued the work and proved the decidability of the class over integers [6]. In [69], we further developed the work of [53] by calculating symbolic (sufficient) conditions for the termination of its subclasses through computer algebra tool, DISCOVERER.

Linear programs with linear ranking functions compose a very small class of programs. As to polynomial programs, Bradley et al in [8] proposed an incomplete method to decide whether a polynomial program terminates by using the technique of finite difference tree. However, the approach can be only used to tackle very simple polynomial programs, that have "polynomial behavior".

Following the research line of [29], Cousot in [16] presented a very general approach to ranking function discovering and invariant generation of linear and polynomial programs. The basic idea of the approach is: First the program semantics is expressed in polynomial form; Then the unknown rank functions and invariants are predefined in parametric form; Consequently, the implication in the Floyd-Hoare-Naur verification conditions is handled by abstraction into numerical constraints by Lagrangian relaxation; Finally, the remaining universal quantification is handled by semidefinite programming and the related solvers. Compared with the approach of [29], Cousot's approach is more efficient, as first-order quantifier elimination is not directly applied there. However, Cousot'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.

2 Semi-algebraic system solving

Let $\mathcal{K}[x_1,...,x_n]$ be the ring of polynomials in n indeterminates with coefficients in the field $\mathcal{K}$ with $X=\{x_1,\cdots,x_n\}$ and the order $x_1\prec x_2\prec\cdots\prec x_n$. Then, the *leading 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 $p=c_mx_k^m+\cdots+c_0$ 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 $c_mx_k^m$ the *leading term* of p w.r.t. x_k and c_m the *leading coefficient*.

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 $CPF(\mathcal{K}[x_1,\ldots,x_n])$ the set of conjunctive polynomial formulae over $\mathcal{K}[x_1,...,x_n]$, which only contain logical connective $\wedge$, respectively.

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

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

A semi-algebraic system (SAS) is a conjunctive polynomial formula of the following form

$$\begin{cases} p_1(\mathbf{u},\mathbf{x})=0,\cdots,p_s(\mathbf{u},\mathbf{x})=0,\\ g_1(\mathbf{u},\mathbf{x})\geq 0,\cdots,g_r(\mathbf{u},\mathbf{x})\geq 0,\\ g_{r+1}(\mathbf{u},\mathbf{x})>0,\cdots,g_t(\mathbf{u},\mathbf{x})>0,\\ h_1(\mathbf{u},\mathbf{x})\neq 0,\cdots,h_m(\mathbf{u},\mathbf{x})\neq 0. \end{cases}$$

It is also written as

$$[F,N,P,H] \tag{1}$$

where F,N,P and H stand for $[p_1,\cdots,p_s]$, $[g_1,\cdots,g_r]$, $[g_{r+1},\cdots,g_t]$ and $[h_1,\cdots,h_m]$, respectively. Herein, $\mathbf{x}=(x_1,\cdots,x_n)$ are variables and $\mathbf{u}=(u_1,\cdots,u_d)$ are parameters whose values are in $\mathbb{R}$ and p_i, g_j, h_k are polynomials in $\mathbb{Q}[\mathbf{u},\mathbf{x}]$ with $n,s\geq 1$, $d,r,t,m\geq 0$.

An SAS is called *constant* if it contains no parameters, *i.e.*, $d=0$, otherwise *parametric*. There are many interesting problems concerning SASs which have important applications. For example, if S is a constant SAS,

1.1 Whether does S have infinite number of real solutions?

1.2 If S has a finite number of (distinct) real solutions, what is the number and how to isolate the real solutions?

if S is a parametric SAS,

2.1 What is the necessary and sufficient condition on the parameters for S to have a prescribed number of (distinct) real solutions?

The problems 1.1 and 1.2 were solved in [58,60]; while the problem 2.1, that is classification of real roots of

PSASs, was well solved in [67]. DISCOVERER [57] is a package of procedures to implement the above theories developed with Maple. Since Maple 13, DISCOVERER has been integrated into the RegularChains package of Maple. The main features of DISCOVERER include *real solution classification* of parametric SASs (i.e. Problem 2.1), and *real solution counting and isolating* of constant SASs (i.e. Problem 1.2).

Let us show by a simple example how to use the function of DISCOVERER in Maple 13. Suppose

$$F = [ax^2 + bx + c], N = [\,], P = [\,], H = [\,],$$

where x is variable and a, b, c are parameters. We want to know the condition on a, b, c such that the system $[F, N, P, H]$ has no real solutions. In Maple 13, we type the following commands.

```
> with(RegularChains):
> with(ParametricSystemTools):
> with(SemiAlgebraicSetTools):
> infolevel[RegularChains]:=1:
> R := PolynomialRing([x, a, b, c]);
> F:=[a*x^2+b*x+c]; N:=[ ]; P:=[ ]; H:=[ ];
> rrc := RealRootClassification(F, N, P, H, 3, 0, R);
```

The first three commands load the package. Although the final result can be read from *rrc*, the fourth command can provide some pretty printing information. The fifth and sixth lines define the system and the last command computes the condition we want. Therein, 3 is the number of parameters and 0 is the prescribed number of real solutions.

For the details of RealRootClassification and RealRootCounting, please refer to the corresponding help pages in Maple 13.

3 Invariant generation and non-linear ranking function discovering

In this section, we will sketch our approach on reduction invariant generation and nonlinear ranking discovering to solving SASs.

3.1 Basic notions

For our purpose, we need the following basic notions.

First, a program is represented as a labeled transition system which 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 transition system* (FPTS)), if all its formulae are in $PF(\mathcal{K}[x_1, \ldots, x_n])$ (resp. $FPF(\mathcal{K}[x_1, \ldots, x_n])$).

A state is a valuation of the variables in V. The state space is denoted by $Val(V)$. Without confusion we will use V to denote both the variable set and an arbitrary state. 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 $l_1 \overset{\rho_\tau, \theta_\tau}{\to} l_2$, or simply by $l_1 \overset{\tau}{\to} l_2$. A sequence of transitions $l_{11} \overset{\tau_1}{\to} l_{12}, \ldots, l_{n1} \overset{\tau_n}{\to} l_{n2}$ is called *composable* if $l_{i2} = l_{(i+1)1}$ for $i = 1, \ldots, n-1$, and written as $l_{11} \overset{\tau_1}{\to} l_{12}(l_{21}) \overset{\tau_2}{\to} \cdots \overset{\tau_n}{\to} l_{n2}$. A composable sequence is a *transition circle* at l_{11}, if $l_{11} = l_{n2}$. For any composable sequence $l_0 \overset{\tau_1}{\to} l_1 \overset{\tau_2}{\to} \cdots \overset{\tau_n}{\to} l_n$, it is easy to show that there is a transition of the form $l_0 \overset{\tau_1;\tau_2;\cdots;\tau_n}{\to} l_n$ 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.

Definition 1 (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 formula ϕ over V such that ϕ holds on all states that can be reached at location l.*

Definition 2 (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 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 $\tau = \langle l_i, l_j, \rho_\tau, \theta_\tau\rangle$,*

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

Definition 3 (Ranking Function) *Assume $P = \langle V, L, T, l_0, \Theta\rangle$ is a transition system. A ranking function is a function $\gamma : Val(V) \to \mathbb{R}^+$ such that the following conditions are satisfied:*

Initiation: $\Theta(V_0) \models \gamma(V_0) \geq 0$.

Decreasing: *There exists $C \in \mathbb{R}^+$ such that for any transition circle $l_0 \overset{\tau_1}{\to} l_1 \overset{\tau_2}{\to} \cdots \overset{\tau_{n-1}}{\to} l_{n-1} \overset{\tau_n}{\to} l_0$ at l_0,*

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

3.2 Generating polynomial invariants

Given an SATS S, the procedure of generating polynomial invariants with the approach of [10] includes the following 4 steps:

1. 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. According to Definition 2, 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 conjunct all the resulting conditions.
3. From Definition 2, 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 $\eta(l_j)$,

$$\eta(l_i) \wedge \rho_\tau \wedge \theta_\tau \wedge \neg\phi \tag{2}$$

   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. 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.

We use the following example first given in [10] and revised in [59] to illustrate the above procedure.

Example 1 *The code of the program is on Fig.1 and its corresponding SATS is on Fig.2.*

$$\begin{array}{ll} & \textbf{\textit{Integer}}\ (x,y) := (0,0); \\ l_0: & \textbf{\textit{while}}\ x \geq 0 \wedge y \geq 0\ \textbf{\textit{do}} \\ & \qquad (x,y) := (x+y^2, y+1); \\ & \textbf{\textit{end while}} \end{array}$$

Fig.1

$$\begin{array}{l} P = \{ \\ \quad V = \{x, y\} \\ \quad L = \{l_0\} \\ \quad T = \{\tau\}\ \ \} \\ \textit{where} \\ \tau = \langle l_0, l_0, x' - x - y^2 = 0 \wedge \\ \qquad\qquad y' - y - 1 = 0, x \geq 0 \wedge y \geq 0\rangle \end{array}$$

Fig.2

Firstly, we predefine a parametric invariant at l_0 as

$$eq(x,y) = a_1y^3 + a_2y^2 + a_3x - a_4y = 0, \tag{3}$$
$$ineq(x,y) = b_1x + b_2y^2 + b_3y + b_4 > 0 \tag{4}$$

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 2, $\Theta \models \eta(l_0)$ is equivalent to neither of the following two PSASs having real solutions.*

$$x = 0, y = 0, eq(x,y) \neq 0 \tag{5}$$
$$x = 0, y = 0, ineq(x,y) \leq 0 \tag{6}$$

Obviously, (5) has no real solutions iff true while (6) has no real solutions iff $b_4 > 0$.

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

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

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

$$eq(x,y) = 0 \wedge x' - x - y^2 = 0 \wedge y' - y - 1 = 0 \wedge \wedge eq(x',y') \neq 0 \tag{8}$$
$$ineq(x,y) > 0 \wedge x' - x - y^2 = 0 \wedge y' - y - 1 = 0 \wedge \wedge ineq(x',y') \leq 0 \tag{9}$$

Setting $R1 := PolynomialRing([y, x, y', x', a_1, a_2, a_3, a_4])$, *by calling*

$$\text{RealRootClassification}([x' - x - y^2, y' - y - 1, eq(x, y)], [\,], [\,], [eq(x', y')\,], 4, 0, R1),$$

we obtain that (8) has no real solutions if and only if

$$a_3y^2 + 3a_1y^2 + 2ya_2 + 3a_1y - a_4 + a_2 + a_1 = 0 \vee \quad (10)$$
$$a_3 = 0. \quad (11)$$

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

$$-a_4 + a_2 + a_1 = 0 \wedge 3a_1 + 2a_2 = 0 \wedge a_3 + 3a_1 = 0, \quad (12)$$

while (11) *leads to a trivial result.*

For (9), setting

$$R2 := PolynomialRing([y, x, y', x', b_1, b_2, b_3, b_4]),$$

by calling

$$\text{RealRootClassification}([x' - x - y^2, y' - y - 1], [-ineq(x', y')], [ineq(x, y)], [\,], 4, 0, R2),$$

we obtain that (9) has no real solutions iff

$$b_4 + b_3 + b_2 + 2b_2y + b_3y + b_2y^2 + b_1x + b_1y^2 > 0. \quad (13)$$

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 and get the following necessary and sufficient condition*

$$b_1 > 0 \wedge b_4 > 0 \wedge b_2 + b_3 + b_4 > 0 \wedge ((b_2 \geq 0 \wedge b_2 + b_3 \geq 0) \vee (b_1 + b_2 \geq 0 \wedge 2b_2 + b_3 \geq 0) \vee d_1 \leq 0 \vee d_2 < 0 \vee (f_1 \leq 0 \wedge f_2 \geq 0)) \quad (14)$$

where

$$\begin{aligned} d_1 &= -4b_1b_3 - 4b_1b_2 + 4b_2^2 \\ d_2 &= -4b_1b_2 - 4b_1b_3 - 4b_4b_1 + b_3^2 - 4b_4b_2 \\ f_1 &= 2b_4b_1^2 + 2b_4b_1b_2 - 2b_2^2b_1 - 4b_1b_2b_3 - b_1b_3^2 + 2b_2^3 \\ f_2 &= b_2^4 - 2b_2^2b_1b_4 + b_4^2b_1^2 - 4b_4b_1b_3b_2 - b_2^2b_3^2 + 4b_2^3b_4 \\ &\quad + b_1b_3^3 + b_1b_2b_3^2 \end{aligned}$$

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

According to (14), it is easy to see that

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

and

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

are invariants of P.

3.3 Discovering non-linear ranking functions

According to the definitions, a ranking function can be seen 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 on discovering non-linear ranking functions in [9] comprises the following 4 steps:

Step 1 Predetermine a template of ranking functions.

Step 2 According to the bounded 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 From Definition 3, there exists a positive constant C such that for any transition circle $l_0 \xrightarrow{\tau_1} l_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_n} l_0$,

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

equivalent to

$$\rho_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \theta_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \gamma(V') < 0, \quad (16)$$
$$\rho_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \theta_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \gamma(V) - \gamma(V') < C \quad (17)$$

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 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.

The above procedure is *complete* also 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.

3.4 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 [21], which is double exponential in 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 generating invariants and ranking functions of program as advocated by Kapur [29].

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 [10], 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.

3.5 Beyond semi-algebraic transition systems

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

3.5.1 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.

We can show that the problem of discovery of invariants and ranking functions of a PTS is equivalent to that of the resulting SATS (the detailed proof can be found in [59]).

3.5.2 Fractional transition systems

According to the results of polynomial algebra and the above discussions, the problems of invariant generation and ranking function discovering for FPTSs can be readily reduced to those of SATSs too.

3.6 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 shown in the previous subsections, we only need to consider the case of SATSs.

3.6.1 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 too, and therefore, we can get another condition only on the parameters. Finally, we can instantiate the parameters according to the resulting condition on them and generate invariants with the predefined template.

3.6.2 Fractional formula as invariant

For this case, in the first step, we predefine a template of invariant that is a parametric fractional formula. According to the results of polynomial algebra, the parametric fractional formula is equivalent to a parametric polynomial formula. So, the rest steps are reduced to those of the above case.

3.6.3 Fractional or radical expression as ranking function

The ranking function is not necessarily a polynomial. By the results of polynomial algebra, similarly to the previ-

ous discussion, it is quite easy to extend the approach of [9] to synthesize a ranking function represented by a fraction.

In addition, Li [34] present another method for the synthesis of nonlinear ranking function of a program loop by employing the region-based search [5]. The idea is to reduce the nonlinear ranking function discovering to the inequality proving. The inequality prover BOTTEMA [66] then can be utilized to check validity for resulting inequalities. In contrast to other approaches, this one can also discover a ranking function with radicals due to BOTTEMA's distinct features.

4 Symbolic decision procedure for termination of linear loops

Tiwari in [53] proved that the termination of the following loops on the reals is decidable,

$$\mathtt{P}_1: \qquad \textbf{while} \quad (Bx > b) \quad \{\, x := Ax + c \,\},$$

where A is an $n \times n$ matrix, B is an $m \times n$ matrix, and x, b and c are vectors. $Bx > b$ is a conjunction of strict linear inequalities which is the loop condition, while $x := Ax + c$ is interpreted as updating the values of x by $Ax + c$ simultaneously and not in any sequential order. We say $\mathtt{P}_1$ terminates if it terminates on all initial values.

The termination problem of $\mathtt{P}_1$ can be easily reduced to that of the following *homogeneous loop*

$$\mathtt{P}_2: \qquad \textbf{while} \quad (Bx > 0) \quad \{\, x := Ax \,\}.$$

A key step of the decision procedure in [53] is to compute the *Jordan form* of the matrix A so that one can have a diagonal description of A^n. In [53] it was proved that if the Jordan form of A is $A^* = Q^{-1}AQ$ and B^* is set to be BQ, then $\mathtt{P}_2$ terminates if and only if

$$\mathtt{P}_2^*: \qquad \textbf{while } (B^*x > 0)\ \{x := A^*x\}$$

terminates. This idea is natural, but, as pointed out in [62], if we use floating-point computation routines to calculate the Jordan form in a conventional way, the errors of floating-point computation may lead to a wrong conclusion. So, symbolic computation is needed in this decision framework. To see this point, let us consider the following homogeneous loop given in [62].

Example 2 *Let*

$$A = \begin{bmatrix} 2 & -3 \\ -1 & 2 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & b \\ -1 & b \end{bmatrix},$$

where $b = -\dfrac{1127637245}{651041667} = -\sqrt{3} + \epsilon \approx -1.732050807$, *with* $\epsilon = \sqrt{3} - \dfrac{1127637245}{651041667} > 0$. *Determine whether or not*

$$\mathtt{Q}_1: \qquad \textbf{while } (Bx > 0)\ \{x := Ax\}$$

is terminating.

According to the conventional method, in order to compute the Jordan form of A we have to calculate the eigenvalues of A by using floating-point computation, say, through the package `linalg` *(or* `LinearAlgebra`*) in Maple 11. The approximate eigenvalues of A are* 3.732050808 *and* 0.267949192 *(both take 10 decimal digits of precision). Hence, the Jordan form of A is*

$$A^* = Q^{-1}AQ = \begin{bmatrix} 3.732050808 & 0 \\ 0 & 0.267949192 \end{bmatrix}$$

where

$$Q = \begin{bmatrix} 0.5 & 0.5 \\ -0.2886751347 & 0.2886751347 \end{bmatrix}$$

Use the same package of Maple 11, we calculate

$$B^* = BQ = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & -1.0 \end{bmatrix}.$$

Then, according to the above discussion, the loop $\mathtt{Q}_1$ is terminating if and only if the following loop $\mathtt{Q}_2$ terminates,

$$\mathtt{Q}_2: \qquad \textbf{while } (B^*x > 0)\ \{x := A^*x\}.$$

Obviously,

$$(A^*)^n = \begin{bmatrix} 3.732050808^n & 0 \\ 0 & 0.267949192^n \end{bmatrix}.$$

If we let $x = [1, -1]^{\mathrm{T}}$, after n times of iteration,

$$(A^*)^n x = [3.732050808^n, -0.267949192^n],$$

where v^{T} stands for the transpose of the vector v. And the loop condition is

$$B^*(A^*)^n x = [3.732050808^n, 0.267949192^n] > [0, 0]$$

which is always true for all n. Therefore, $\mathtt{Q}_1$ is not terminating.

However, this conclusion is not correct. Let us see how the floating-point computation leads us to the wrong result. The Jordan form of A is indeed (by symbolic computation)

$$J = P^{-1}AP = \begin{bmatrix} 2+\sqrt{3} & 0 \\ 0 & 2-\sqrt{3} \end{bmatrix},$$

where

$$P = \begin{bmatrix} \dfrac{1}{2} & \dfrac{1}{2} \\ -\dfrac{1}{6}\sqrt{3} & \dfrac{1}{6}\sqrt{3} \end{bmatrix}$$

and, in order to obtain B^*, we should compute BP instead of BQ. Symbolically,

$$BP = \begin{bmatrix} \frac{1}{2} - \frac{b}{6}\sqrt{3} & \frac{1}{2} + \frac{b}{6}\sqrt{3} \\ -\frac{1}{2} - \frac{b}{6}\sqrt{3} & -\frac{1}{2} + \frac{b}{6}\sqrt{3} \end{bmatrix} = \begin{bmatrix} 1 - \frac{\epsilon}{6}\sqrt{3} & \frac{\epsilon}{6}\sqrt{3} \\ -\frac{\epsilon}{6}\sqrt{3} & -1 + \frac{\epsilon}{6}\sqrt{3} \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}.$$

Therefore $m_{12} > 0$, $m_{21} < 0$. However, when we use floating-point computation, these two entries (m_{12} and m_{21}) are evaluated to 0 (in Maple 11 with Digits 10). That is why we obtain wrong result by floating-point computation.

One may guess that if we evaluate BP rather than BQ through floating-point computation routines, we may obtain a more precise approximation. Unfortunately it is not true. In fact using floating-point computation to compute BP we will still get some strange results. For example, Maple 11 with Digits 10 gives BP the following matrix

$$\begin{bmatrix} 1.0 & -1.0 \cdot 10^{-10} \\ 1.0 \cdot 10^{-10} & -1.0 \end{bmatrix}$$

which is totally wrong, because, comparing with the signs of m_{12} and m_{21} in the above, m_{12} is negative and m_{21} positive.

To handle the above problem, in [62], we developed a symbolic decision procedure for the termination of linear programs $\mathtt{P}_1$. The general framework of our procedure is quite similar to that of Tiwari's [53], but we re-implement the two key steps, i.e. computing Jordan normal form of A and generating linear constraints, based on symbolic computation. Thus, our decision procedure avoid errors caused by floating-point computation.

The basic idea of our approach is: Firstly, represent eigenvalues of A symbolically, and then symbolically compute a set of eigenvectors and generalized eigenvectors of A, which can form an invertible matrix P such that $P^{-1}AP$ is the Jordan normal form of A. Thus, the termination problem of $\mathtt{P}_2$ is symbolically reduced to that of $\mathtt{P}_2^*$. Secondly, we presented a symbolic decision procedure to determine if $\exists x \forall n \in \mathbb{N}. B^*(A^*)^n x > 0$, i.e. whether or not P_2^* terminates. Furthermore, by complexity analysis, we showed that our symbolic decision procedure is as efficient as the one given in [53]. The complexity of our algorithm is $\max\{O(n^6), O(n^{m+3})\}$, where n is the number of variables of $\mathtt{P}_2$ and m is the number of the Boolean conditions of $\mathtt{P}_2$. In contrast, the complexity of the decision procedure developed in [53] is $O(n^{m+3})$.

In addition, in [62] we also considered the case when the characteristic polynomial of A is irreducible. A much simpler and more efficient decision algorithm was invented by reduction to solving a semi-algebraic system only with one variable. The complexity of the algorithm for this case is $\max\{O(n^6), O(mn^3)\}$.

Bi and Shan [2] proposed a heuristic method to analyze the termination of a linear loop program with conditionals, that is, with "if...else..." statement. The method first transforms a linear loop program with conditionals into a nested linear loop program and then checks whether or not each of the inner loops terminates making use of positive eigenvalues and the corresponding eigenvectors. If one of the inner loops in the nest linear loop is nonterminating, then, so is the original linear loop. Otherwise, it still uses ranking functions by quantifier elimination to analyze the termination.

5 Termination analysis of nonlinear loops

For a generic loop

$$\textbf{while} \quad (\mathit{constraints}) \quad \{\mathit{updates}\},$$

it is well known that the termination problem is undecidable in general, even for a simple class of polynomial programs [8]. In [4] Blondel et al. proved that, even when all the constraints and updates are given as piecewise linear functions, the termination of the loop remains undecidable.

If the constraints or the updating function in the above loop is not linear, the loop is called *nonlinear*. In this section, we shall introduce briefly two recent decision results on termination of some classes of nonlinear loops.

5.1 Linear programs with nonlinear constraints

In this subsection, we introduce the result of [61]. Let us consider the problem of termination of the following loop:

$$\tilde{\mathcal{P}}_1: \qquad \textbf{while} \quad (P(X) > b) \quad \{X := AX + C\},$$

where $X = [x_1 \ \ldots \ x_N]^{\mathrm{T}}$ is the vector of state variables of the program, $P(X) = [P_1(X) \ \ldots \ P_M(X)]^{\mathrm{T}} > b$ are polynomial constraints, each $P_i(X)(1 \le i \le M)$ is a polynomial in $\mathbb{Q}[X]$ and A is an $N \times N$ matrix over $\mathbb{Q}$ (the rational numbers). That is to say, we replace the linear constraints in $\mathtt{P}_1$ with polynomial constraints and keep linear updates unchanged. If $b = \mathbf{0}$ and $C = \mathbf{0}$, $\tilde{\mathcal{P}}_1$ becomes

$$\mathcal{P}_1: \qquad \textbf{while} \quad (P(X) > 0) \quad \{X := AX\}$$

which is called the *homogeneous* case of $\tilde{\mathcal{P}}_1$. It is easy to show that the termination of $\tilde{\mathcal{P}}_1$ can be reduced to that of some $\mathcal{P}_1$. Without loss of generality, we concentrate on the termination of $\mathcal{P}_1$.

The main contributions of [61] are as follows. First, it is proven that the termination of $\mathcal{P}_1$ over $\mathbb{Z}$ is undecidable by reduction to Hilbert's 10^{th} problem. Moreover,

Xia and Zhang proved that, if "$>$" is replaced with "$\geq$" in $\mathcal{P}_1$, the termination of the resulted $\mathcal{P}_1$ over $\mathbb{Z}$ is still undecidable. Second, an algorithm is proposed to decide the termination of $\mathcal{P}_1$ under a reasonable assumption.

Let us explain the basic idea of the algorithm. To decide whether $\mathcal{P}_1$ is terminating, it is equivalent to check whether there exists X such that $P(A^nX) > 0$ holds for all $n \geq 1$. That means one should check the signs of each $P_i(A^nX)$ for all $n \geq 1$. So, it is natural to compute first a general expression for each $P_i(A^nX)$ of $P(A^nX)$ for any n. To this end, Xia and Zhang in [61] first computed A^nX, the value of state variables X after n iterations, by means of generating functions and obtained a unified formula expressing each entry of A^nX. The method is more convenient for subsequent computation than the one using Jordan form.

Then the value of $P_i(A^nX)$ was expressed with the formula. The expression of $P_i(A^nX)$ is much more complicated than that of linear case. To determine whether $P_i(A^nX) > 0$ as $n \to +\infty$, it is sufficient to check the sign of coefficient of its *dominant term* as $n \to +\infty$. So, a suitable order was introduced so that one can decide which term is the *dominant* (*leading*) term.

The algorithm proposed in [61] first guesses a leading term for each expression and sets its coefficient to be "positive" and the coefficients of the terms with higher order to be "nonnegative". This constructs a semi-algebraic system (SAS). If a guess is satisfiable, *i.e.*, the corresponding SAS has solutions, the algorithm returns nonterminating. Otherwise, it returns terminating.

The algorithm is incomplete in the sense that if it returns nonterminating, $\mathcal{P}_1$ is indeed nonterminating, but if it returns terminating, $\mathcal{P}_1$ may not terminate. However, if $\mathcal{P}_1$ has a special property, the algorithm is complete. We do not want to explain the property here since it involves many technical details. We only comment that a large subclasses of $\mathcal{P}_1$ satisfy the property. Finally, it is conjectured that the termination of $\mathcal{P}_1$ over $\mathbb{R}$ is undecidable in general.

Moreover, Wu, Shen, Bi and Zeng [55,56] considered the termination of linear loops with some more polynomial guards different from the above one. The termination decision is also reduced to semi-algebraic system solving by symbolically analyzing the Jordan form of the assignment matrix.

5.2 Nonlinear loops over intervals

In this subsection, we introduce the main result of [70] obtained by Yao. The following loop is called *loop over intervals*

$$\mathcal{P}_\Omega: \quad \textbf{while} \quad (x \in \Omega) \quad \{x := f(x)\}$$

where x is the only program variable, Ω is an interval and f is a continuous function. Yao in [70] gives some surprisingly simple decision results on the termination of such loops.

Firstly, he proved that if $\Omega = (a, b) \subseteq \mathbb{R}$ and $\mathcal{P}_\Omega$ is nonterminating, then f must have a fixed point in $[a, b] \subseteq \mathbb{R}_\infty$, where $\mathbb{R}_\infty = \mathbb{R} \cup \{+\infty, -\infty\}$. An immediate corollary is that if $\Omega = [a, b]$, then $\mathcal{P}_\Omega$ is nonterminating if and only if f has a fixed point in Ω. That means, in this case, one only needs to check whether $f(x) = x$ is true on Ω.

Secondly, he further proved that if f only has fixed point(s) at the endpoint(s) of Ω and Ω itself is not a closed interval, then the decision result on the termination of $\mathcal{P}_\Omega$ is also obtained by requiring f to have some additional properties. These properties are satisfied by a large subclasses of continuous functions. For example, piecewise polynomial functions satisfy these properties. Yao in [70] also proposed a decision algorithm for the termination of $\mathcal{P}_\Omega$ when f is a piecewise polynomial function.

As pointed out in [70], a valuable future work along this direction is how to generalize those results to the cases when Ω is the union of intervals or the program has more than one variables.

5.3 P-solvable loops without conditionals

Bi, Shan and Wu [3] presented an algorithm to analyze termination of a class of nonlinear loop programs called *P-solveble* loop programs [31] without " if...else..." statements. This algorithm is based on recurrence solving and quantifier elimination.

6 Conclusion and Future Work

In this paper, we summarized our recent research on program verification through computer algebra. The main results are:

- A complete approach on automatic discovery of invariants and ranking functions by reduction to semi-algebraic systems solving. Compared with well-established work in this field, the merits of our approach are: First, using it one can more efficiently generate more expressive invariants; Second, it can be applied to nonlinear programs and discovering nonlinear ranking functions, in contract that well-known approaches can only applied to linear programs and discovering linear ranking functions; Finally, it is also complete.
- A symbolic decision procedure for a class of simple linear loops first investigated in [53]. Our decision procedure is as efficient as the one given in [53] based on numerical computation, but our decision procedure can avoid the incorrectness caused by rounding error of floating point computation. In addition, for

the case when the characteristic formula of the assignment matrix of a program is irreducible, we presented a very fast algorithm to decide if the program terminates or not.
- The termination analysis of programs, in particular non-linear programs.

Our ongoing and future work include

- How to verify programs with complicated data structures is a big challenge. All well-established work mainly focus on simple data, in particular numbers. In practice, most programs contain complicated data structures, such as array, pointer, record, etc. So far, little success has been made towards the problem. The potential solution could be to invent more theories for specific data structures and combine different theories together.
- It deserves to consider the verification of numerical programs, in particular backwards error analysis.
- It would be interesting to investigate more specific classes of programs that have important applications in practice and find out decision procedures for their termination problems. We also need to invent more practical, but may be not complete termination analysis methods.
- It deserves to consider how to apply the techniques and tools on computer algebra to verification of hybrid systems, in particular nonlinear hybrid systems.
- We are interested in carrying out more case studies under different program paradigms to understand the advantage as well as the limitation of computer algebra tools.

Acknowledgements

We would like to thank our colleagues Prof. Zhi-Bin Li, Prof. Zhenbing Zeng, Dr. Liyong Shen, Dr. Min Wu, Dr. Shizhong Zhao, Dr. Zhongqing Bi, Dr. Liangyu Chen, Dr. Yi Li, Dr. Meijing Shan, Dr. Ming Xu, Dr. Bin Wu, Mr. Yinghua Chen, Mr. Yong Yao, Mr. Zhihai Zhang, and all the students involving in this research for their contributions to the projects.

This work was initiated by the project "Applying Computer Algebra to Formal Methods" under the grant 60573007 funded by NSFC, and then further stressed by the key project "Computer Algebra based Analysis, Verification and Tools for Embedded Systems" under the grant 90718041 within the framework of "Trustworthy Software Plan" funded by NSFC. The first and last authors are partly supported by the grants NKBRFC-2004CB318003 and NKBRFC-2005CB321902, and the second and third authors are partly supported by the grants NSFC-60721061 and NSFC-60736017.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(3):3-34, 1995.
2. Z. Bi and M. Shan. Termination analysis of linear programs with conditionals. International Conference on Advanced Computer Theory and Engineering 2008, Phuket, Thailand. 2008: 450-456.
3. Z. Bi, M. Shan and B. Wu. Termination analysis of P-solvable loops with assignments only. 2008 International Symposium on Information Science and Engineering, Shanghai, China. 2008: 125-129.
4. V.D. Blondel, O. Bournez, P. Koiran, C.H. Papadimitriou, J.N. Tsitsiklis. Deciding stability and mortality of piecewise affine dynamical systems. *Theoretical Computer Science*, 2001, 255(1–2): 687–696
5. A. Bradley, Z. Manna and H. Sipma. Termination analysis of integer linear loops. In LNCS, vol, 3653, pp. 488-502, Springer, Heidelberg, 2005.
6. M. Braverman. Termination of integer linear programs. In Proc. CAV'06, LNCS 4144, pp. 372-385. 2006.
7. F. Besson, T. Jensen, and J.-P. Talpin. Polyhedral analysis of synchronous languages. In SAS'99, LNCS 1694, pp. 51-69, Springer-Verlag, 1999.
8. A. Bradley, Z. Manna and H. Sipma. Termination of polynomial programs. In Proc. of VMCAI'05, LNCS
9. Y. Chen, B. Xia, L. Yang, N. Zhan and C. Zhou. Discovering non-linear ranking functions by solving semi-algebraic systems. In proc. ICTAC'07, LNCS 4711, pp.34-49.
10. 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.
11. E.M. Clarke and E.A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic, *in IBM Workshop on Logics of Programs*, LNCS 131, pp.52-71, 1981.
12. G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. *Automata Theory and Formal Languages*, LNCS 33, pp. 134–183, 1975.
13. G. E., Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. of Symbolic Computation*, 12:299–328, 1991.
14. 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.
15. M. Colón and H.B. Sipma. Synthesis of linear ranking functions. In TACAS'01, LNCS 2031, pp. 67–81, 2001.
16. P. Cousot. Proving program invariance and termination by parametric abstraction, Langrangian Relaxation and semidefinite programming. In VMCAI 2005, LNCS 3385, pp. 1-24. 2005.
17. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among the variables of a program. In ACM POPL'78, pp. 84-97, 1978.
18. P. Cousot and R. Cousot. Abstraction interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In ACM POPL'77, pp. 238–252, 1977.

19. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In ACM POPL'79, pp. 269-282, 1979.
20. 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.
21. J. H., Davenport and J. Heintz. Real elimination is doubly exponential. *J. of Symbolic Computation,* 5:29–37, 1988.
22. R. W. Floyd. Assigning meanings to programs. In *Proc. Symphosia in Applied Mathematics 19*, pp. 19-37, 1967.
23. S. German and B. Wegbreit. A synthesizer of inductive assertions. *IEEE Transactions on Software Engineering* 1(1):68-75. 1975.
24. D. Harel and A. Pnueli. On the development of reactive systems. In K.R. Apt, editor, *Logics and Models of Concurrency Systems,* volume 13 of NATO, ASI Series, pp. 447-498. Springer-Verlag, New York, 1985.
25. T. A. Henzinger and H. Wong-Toi. Linear phase-portrait approximations for nonlinear hybrid systems. In Hybrid Systems'95, LNCS 1066, pp.377-388.
26. C.A.R. Hoare. An axiomatic basis for computer programming. *Comm. ACM*, 12(10):576-580. 1969.
27. M. Karr. Affine relationships among variables of a program. *Acta Informatica* 6:133-151. 1976.
28. S. Katz and Z. Manna. Logical analysis of programms. *Communications of the ACM* 19(4):188-206. 1976.
29. 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.
30. Y. Kesten, and A. Pnueli. Timed and hybrid statecharts and their textual representation. In J. Vytopil, editor, Formal Techniques in Real-Time and Fault-Tolerant Systems, volume 571 of Lecture Notes in Computer Science, pages 591-619. 1992.
31. L. Kovács. *Automated Invariant Ceneration by Algebraic Techniques for Imperative Program Verification in Theorema.* PhD thesis, Hohannes Kepler University of Linz, 2007.
32. L. Lamport. Hybrid systems in TLA+. In A. Nerode and A. P. Ravn and H. Rischel, editors, Hybrid Systems'92, volume 736 of Lecture Notes in Computer Science, pp. 77-102. 1992.
33. G. Lafferrierre, G.J. Pappas and S. Yovine. Symbolic reachability computaion for families of linear vector fields. *J. of Symbolic Computation* 11:1–23, 2001.
34. Y. Li. Automatic discovery of non-linear ranking function of loop programs. ICCSIT'09, vol. 1, pp, 402-406.
35. J. L. Lions. The ARIANE 5 Flight 501 Failure Report. 19, July 1996, *European Space Agency (ESA).*
36. Z. Manna, and A. Pnueli. Verifying hybrid systems. In R.L. Grossman, A. Nerode, A. Ravn, and H. Rischel, editors, Hybrid Systems, volume 736 of Lecture Notes in Computer Science, pages 4-35. 1993.
37. B. Moszkowski. A temporal logic for multilevel reasoning about hardware. *IEEE Computers*, 18(2):10-19, 1985.
38. M. MÍźller-Olm and H. Seidl. Polynomial constants are decidable. 9th Static Analysis Symposium (SAS'02), LNCS 2477, pp. 4-19. 2002.
39. M. MÍźller-Olm and H. Seidl. Precise interprocedural analysis through linear algebra. ACM SIGPLAN Principles of Programming Languages, POPL 2004, pp. 330-341. 2004.
40. P. Naur. Proofs of algorithms by general snapshops. *BIT*, 6:310-316, 1966.
41. T. Nipkow, C.P. Paulson and M. Wenzel. *Tutorial on ISABELLE/HOL.* Springer-Verlag, 2009.
42. S. Owre, J.M. Rushby and N. Shankar. PVS: A protype verification system. In CADE'92, LNCS 607, pp. 748-752, 1992.
43. C. Paulin-Mohring and B. Werner. Synthesis of ML programs in the system Coq. *J. Symbolic Logic,* 15(5/6):607-640, 1993.
44. A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In VMCAI'04, LNCS 2937, pp. 239–251, 2004.
45. J.-P. Queille and J. Sifakis. Verification of concurrent systems in CESAR. In Int. Symp. On Programming, LNCS 137, pp.337-351, 1982.
46. G. Reeves and T. Neilson. The Mars Rover Spirit FLASH anomaly. *IEEE Aerospace Conference'05.*
47. 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.
48. 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.
49. E. Rodriguez-Carbonell and D. Kapur. Generating all polynomial invariants in simple loops. *Journal of Symbolic Computation*, 42:443-476. 2007.
50. S. Sankaranarayanan, H.B. Sipma, and Z. Manna. Nonlinear loop invariant generation using Gröbner bases. In ACM POPL'04, pp. 318–329, 2004.
51. R. Skeel. Roundoff error and the Patriot missile. *SIAM News,* **25(4)**, 11(1992).
52. *International Conference on Verified Software: Theories, Tools and Experiments*, ETH Zürich, Oct. 10-13, 2005.
53. A. Tiwari. Termination of linear programs. In CAV'04, LNCS 3114, pp. 70–82, 2004.
54. B. Wegbreit. The synthesis of loop predicates. *Communications of the ACM* 17(2):102-112. 1974.
55. B. Wu, L. Shen, Z. Bi and Z. Zeng. Termination of loop programs with polynomial guards. (submitted)
56. B. Wu, L. Shen, Z. Bi and Z. Zeng. Termination of a class of the programs with polynomial guards. International Conference on Information Management and Engineering, IEEE Computer Society, pages 274-277, 2009.
57. B. Xia. DISCOVERER: A tool for solving semi-algebraic systems. Software Demo at ISSAC 2007, Waterloo, 2007. Also: *ACM SIGSAM Bulletin*, 2007, 41(3): 102–103
58. B. Xia and L. Yang. An algorithm for isolating the real solutions of semi-algebraic systems. *J. Symbolic Computation,* 34:461–477, 2002.
59. B. Xia, L. Yang and N. Zhan. Program verification by reduction to semi-algebraic systems solving. In Proc. of ISoLA'08, CCIS 17 (T. Margaria and B. Ste en Eds.), Springer-Verlag, pp277-291, 2008.

60. B. Xia and T. Zhang. Real solution isolation using interval arithmetic, *Comput. Math. Appl.*, vol. 52, pp.853–860, 2006.
61. B. Xia and Z. Zhang. Termination of linear programs with nonlinear constraints. arXiv:0904.3588v1, April 2009.
62. B. Xia, L. Yang, N. Zhan, Z. Zhang. Symbolic decision procedure for termination of linear programs. (DOI:10.1007/s00165-009-0144-5). To appear in *Formal Aspects of Computing*.
63. M. Xu, L. Chen and Z.-B. Li. Reachability computation of a class of nonlinear systems. In *Proc. of International Conference on Computer and Information Science 2009*, pp.706-710. IEEE Computer Society.
64. M. Xu, L. Chen, Z. Zeng, and Z.-B. Li. Reachability analysis of rational eigenvalue linear systems. To appear in *International Journal of Systems Science*.
65. L. Yang. Recent advances on determining the number of real roots of parametric polynomials. *J. Symbolic Computation*, 28:225–242, 1999.
66. L. Yang. Recent advances in automated theorem proving on inequalities . *J. of Computer Science and Technology*, 14(5) pp. 434–446, 1999.
67. L. Yang, X. Hou and B. Xia. A complete algorithm for automated discovering of a class of inequality-type theorems. *Sci. in China (Ser. F)* 44: 33–49 (2001).
68. L. Yang, X. Hou and Z. Zeng. A complete discrimination system for polynomials. *Science in China (Ser. E)*, 39:628–646, 1996.
69. L. Yang, N. Zhan, B. Xia and C. Zhou. Program verification by using DISCOVERER. In Proc. VSTTE'05, LNCS 4171, pp.575-586.
70. Y. Yao. Termination of nonlinear programs over intervals (in Chinese). To appear in *Journal of Software*.
71. C. Zhou, C.A.R. Hoare and A. Ravn. A calculus of durations. *Inf. Processing Letters*, 40(5):269-276, 1991.