Discovering Non-linear Ranking Functions by Solving Semi-algebraic Systems*

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

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

Abstract. Differing from [6] this paper reduces non-linear ranking function discovering for polynomial programs to semi-algebraic system solving, and demonstrates how to apply the symbolic computation tools, DISCOVERER and QEPCAD, to some interesting examples.

Keywords: Program Verification, Loop Termination, Ranking Function, Polynomial Programs, Semi-Algebraic Systems, Computer Algebra, DISCOVERER, QEPCAD.

1 Introduction

The design of reliable software is a grand challenge in computer science [17] in the 21st century, as our modern life becomes more and more computerized. One of the bases for designing reliable software is the correctness of programs. The dominant approach to automatic program verification is the so-called *Floyd-Hoare-Dijkstra's inductive assertion method* [8,10,11], by using *pre-* and *post-* conditions, loop invariants and proving loop termination 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.

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

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

** The corresponding author: South Fourth Street, No. 4, Zhong Guan Cun, Beijing, 100080, P.R. China, znj@ios.ac.cn

C.B. Jones, Z. Liu, J. Woodcock (Eds.): ICTAC 2007, LNCS 4711, pp. 34–49, 2007.

[7,5,13]. [7] 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, [5] utilized the theory of polyhedra to synthesize linear ranking function of linear programs, and [13] first presented a complete method to find out linear ranking functions for a class of linear programs 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 eventually 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 have no ranking functions. Furthermore, even if a (linear) program has ranking functions, it may not have a linear ranking function. we will show this point by an example later in the paper. Besides, it is well-known that the termination of programs is undecidable in general, even for the class of linear programs [16] or a simple class of polynomial programs [2]. In contrast to the above approach, [16,1] tried to identify decidable subclasses and proved the decidability of the termination problem for a special class of linear programs over reals and integers, respectively. [27] further developed the work of [16] 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, [2] proposed an incomplete method to decide whether a polynomial program terminates by using the technique of finite difference tree. However, [2] can only tackle very simple polynomial programs, that have 'polynomial behaviour'.

In 2005, [6] presented a very general approach to ranking function discovery as well as invariant generation of polynomial programs by parametric abstraction, Lagrangian relaxation and semidefinite programming. The basic idea of the approach is: first the program semantics is expressed in polynomial form; then the unknown ranking function and invariants are abstracted in parametric form; the verification conditions are abstracted as numerical constraints of the parameters through Lagrangian relaxation; the remaining universal quantifications are handled by semidefinite programming; finally the parameters are computed using semidefinite programming solvers. [6] does not directly use the first-order quantifier elimination method due to its bad complexity of doubly exponential[1]. Therefore the approach of [6] is incomplete in the sense that for some program that may have ranking functions and invariants of the predefined form, however applying the approach cannot find them, as Lagrangian relaxation and over-approximation of the positive semi-definiteness of a polynomial are applied.

We here use *semi-algebraic transition system* (SATS), which is an extension of algebraic transition systems in [14], to represent polynomial programs. Then, for a loop in a given SATS (i.e. a given polynomial program), we can first assume

[1] [6] does not provide information about complexity of its own approach. But we are afraid that the complexity of the semidefinite programming, in particular, when the Gram Matrix method is involved, is also bad.

to be a ranking function a polynomial in its program variables with parametric coefficients. In order to determine the parameters, we translate the definition of ranking functions in regard with the predefined polynomial into several SASs, and prove that the polynomial is a ranking function of the loop if and only if each of these SASs has no real solutions. After the translation we apply the functions of root classification for parametric SASs [22,23,24] (and real root isolation for constant SASs [20] if needed) of DISCOVERER to each of these SASs to generate conditions on the parameters. If some universally quantified program variables remain in the resulted conditions, in order to acquire a condition only on the parameters, we can apply QEPCAD to eliminate the remaining quantifiers. In case that the final condition on parameters is still very complicated, applying PCAD (partial cylindrical algebra decomposition [4]) included in both DISCOVERER and QEPCAD, we can conclude whether the condition can be satisfied. If yes, we can further apply PCAD to get the instantiations of these parameters, and therefore achieve a specific polynomial ranking function as necessary. If not, we can define another polynomial template and repeat the above procedure. So our approach does not compromise its completeness. As to the complexity of the approach, DISCOVERER functions on root classification and real root isolation for SASs include algorithms, such as Wu's triangularization [18], to eliminate variables through equalities in SASs with a cost of singly exponential in the number of variables and parameters. Hence, the application of these algorithms can dramatically ease the application of other algorithms of DISCOVERER, QEPCAD and/or PCAD, although they still cost doubly exponential but in the number of *remaining* variables and parameters. A detailed analysis of the complexity will be published in a later paper. In this paper, by briefing the theories behind DISCOVERER we show why the tool works and by applying to examples we show how the tool discovers ranking functions for polynomial programs.

The rest of this paper is structured as follows: Section 2 presents a brief review of the theories and tools of semi-algebraic systems, in particular, the theories on root classification of parametric semi-algebraic systems and on real root isolation of constant SASs, and their implementations in the computer algebra tool DISCOVERER; We extend the notion of algebraic transition systems of [14] to semi-algebraic transition system to represent polynomial programs in Section 3; In Section 4, we use examples to illustrate the reduction of non-linear ranking function discovering to SAS solving; and Section 5 draws a summary and discusses future work.

2 Theories and Tools on Solving Semi-algebraic Systems

In this section, we introduce the theories and the tool DISCOVERER [2] on solving SASs.

[2] DISCOVERER can be downloaded at http://www.is.pku.edu.cn/∼xbc/ discoverer.html

2.1 Semi-algebraic Systems

Let $\mathcal{K}$ be a field, $X = \{x_1, \cdots, x_n\}$ a set of indeterminates, and $\mathcal{K}[x_1, ..., x_n]$ the ring of polynomials in the n indeterminates with coefficients in $\mathcal{K}$, ranged over $p(x_1, \ldots, x_n)$ with possible subscription and superscription. Let the variables be ordered as $x_1 \prec x_2 \prec \cdots \prec x_n$. Then, the *leading variable* of a polynomial p is the variable with the biggest index which indeed occurs in p. If the leading variable of a polynomial p is x_k, p can be collected w.r.t its leading variable as $p = c_m x_k^m + \cdots + c_0$ 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 $p(x_1, \ldots, x_5) = x_2^6 x_3 + 2x_1^4 x_4^4 + (3x_2 x_3 + x_1)x_4^5$, so, its leading variable, term and coefficient are x_4, $(3x_2 x_3 + x_1)x_4^5$ and $3x_2 x_3 + x_1$, respectively.

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

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

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

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

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

where $r > 1, t \geq k \geq 0, m \geq 0$ 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 $d \neq 0$, otherwise* constant.

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

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

2.2 Real Solution Classification

In this subsection, we give a sketch of our theory for real root classification of parametric SASs. For details, please be referred to [23,26].

A finite set of polynomials $\mathbb{T}: [T_1, ..., T_k]$ is called a *triangular set* if it is in the following form

$$\begin{aligned} &T_1 = T_1(x_1, ..., x_{i_1}),\\ &T_2 = T_2(x_1, ..., x_{i_1}, ..., x_{i_2}),\\ &\cdots\cdots\\ &T_k = T_k(x_1, ..., x_{i_1}, ..., x_{i_2}, ..., x_{i_k}), \end{aligned}$$

where x_i is the leading variable of T_i and $x_1 \preceq x_{i_1} \prec x_{i_2} \prec \cdots \prec x_{i_k} \preceq x_s$. For any given SAS S in the form of (1), where the variables in the order $x_1 \prec \cdots \prec x_s$ and all polynomials in S are viewed as polynomials in $\mathbb{Q}(\mathbf{u})[\mathbf{x}]$, we first decompose the equations in S into triangular sets, that is, we transform the polynomial set $\mathbb{P} = [p_1, ..., p_r]$ into a finite set $\mathcal{T} = \{\mathbb{T}_1, ..., \mathbb{T}_e\}$ where each $\mathbb{T}_i$ is a triangular set. Furthermore, this decomposition satisfies $\text{Zero}(\mathbb{P}) = \bigcup_{i=1}^{e} \text{Zero}(\mathbb{T}_i/J_i)$, where $\text{Zero}(\mathbb{P})$ denotes the set of *common zeros* (in some extension of the field of rational numbers) of $p_1, ..., p_r$, $\text{Zero}(\mathbb{T}_i/J_i) = \text{Zero}(\mathbb{T}_i) \setminus \text{Zero}(\{J_i\})$, where J_i is the product of *leading coefficients* of the polynomials in $\mathbb{T}_i$ for each i. It is well-known that the decomposition can be realized by some triangularization methods such as Wu's method [18].

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

$$p_1 = x^2 + y^2 - xy - 1,\ p_2 = y^2 + z^2 - yz - a^2,\ p_3 = z^2 + x^2 - zx - 1,$$

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

$$\begin{aligned} &\mathbb{T}_1 : [x^2 - ax + a^2 - 1, y - a, z - a],\\ &\mathbb{T}_2 : [x^2 + ax + a^2 - 1, y + a, z + a],\\ &\mathbb{T}_3 : [2x^2 + a^2 - 3, 2y(x - y) + a^2 - 1, (x - y)z + xy - 1]. \end{aligned}$$

To simplify our description, let us suppose the number of polynomials in each triangular set is equal to the number of variables as in the above example. For discussion on the other cases of $\mathcal{T}$, please be referred to [19]. That is to say, we now only consider triangular system

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

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

$$F = a_0x^m + a_1x^{m-1} + \cdots + a_{m-1}x + a_m,\ G = b_0x^l + b_1x^{l-1} + \cdots + b_{l-1}x + b_l.$$

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

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

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

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

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

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

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

Sometimes, with a little abuse of notation, we also use BP to denote the square-free part or the set of square-free factors of BP.

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

$$BP = a(a-1)(a+1)(a^2-3)(3a^2-4)(3a^2-1).$$

From the result in [23,26], we may assume $BP \not\equiv 0$. In fact, if any factor of BP is a zero polynomial, we can further decompose the system into new systems with such a property. For a parametric SAS, its border polynomial is a polynomial in the parameters with the following property.

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

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

Example 3. For the system RS in Example 1, $BP = 0$ gives $a = 0, \pm 1, \pm \frac{\sqrt{3}}{3}, \pm \frac{2\sqrt{3}}{3}, \pm \sqrt{3}$. The reals are divided into several open intervals by these points. Because $a \geq 1$, we only need to choose, for example, $9/8, 3/2$ and 2 from $(1, \frac{\sqrt{3}}{3}), (\frac{\sqrt{3}}{3}, \frac{2\sqrt{3}}{3})$ and $(\frac{2\sqrt{3}}{3}, \sqrt{3})$, respectively. Then, we substitute each of the three values for a in the system, and compute the number of distinct real solutions of the system, consequently obtain the system has respectively $8, 4$ and 0 distinct real solutions.

The above three steps constitute the main part of the algorithm in [23,26,19], which, for any input SAS S, outputs the so-called border polynomial BP and a quantifier-free formula Ψ in terms of polynomials in parameters $\mathbf{u}$ (and possible some variables) such that, provided $BP \neq 0$, Ψ is the necessary and sufficient condition for S to have the given number (possibly infinite) of real solutions.

Finally, if we want to discuss the case when parameters are on the "boundary" $BP = 0$, we put $BP = 0$ (or some of its factors) into the system and apply a similar procedure to handle the new SAS.

Example 4. By the steps described above, we obtain the necessary and sufficient condition for RS to have 4 distinct real solutions is $3a^2 - 4 > 0 \ \wedge \ a^2 - 3 < 0$ provided $BP \neq 0$. Now, if $3a^2 - 4 = 0$, adding the equation into the system, we obtain a new SAS $[\ [3a^2 - 4, p_1, p_2, p_3], [a - 1], [\ ], [\ ]\ ]$. By the algorithm in [20,21], we know the number of distinct real solutions of the system is 6.

2.3 DISCOVERER

In this section, we will give a short description of the main functions of DISCOVERER which includes an implementation of the algorithms presented in the previous subsection with Maple. The reader can refer to [23,26] for details. The prerequisite to run the package is Maple 7.0 or a later version of it.

The main features of DISCOVERER include

Real Solution Classification of Parametric Semi-algebraic Systems

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

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

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

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

and results in the necessary and sufficient condition as well as the *border polynomial* BP of T in u such that the number of the distinct real solutions of T exactly equals to N or belongs to N provided $BP \neq 0$. If T has infinite

real solutions for generic value of parameters, BP may have some variables. Then, for the "boundaries" produced by "**tofind**", i.e. $BP = 0$, we can call

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

to obtain some further conditions on the parameters.

Real Solution Isolation of Constant Semi-algebraic Systems

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

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

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

Comparing with other well-known computer algebra tools like REDLOG [9] and QEPCAD [4], DISCOVERER has distinct features on solving problems related to root classification and isolation of SASs through the *complete discrimination system* [24].

3 Polynomial Programs

A polynomial program takes polynomials of $\mathbb{R}[x_1, \ldots, x_n]$ as its only expressions, where $x_1, \ldots, x_n$ stands for the variables of the program. Polynomial programs include expressive class of loops that deserves a careful analysis.

For technical reason, similar to [14], we use algebraic transition systems (ATSs) to represent polynomial programs. An ATS is a special case of standard transition system, in which the initial condition and all transitions are specified in terms of polynomial equations. The class of polynomial programs considered in this paper is more general than the one given in [14] by allowing each assignment inside a loop body to have a guard and its initial and loop conditions possibly with polynomial inequalities. We therefore accordingly extend the notion of algebraic transition systems in [14] by associating with each transition a conjunctive polynomial formula as guard and allowing the initial condition possibly to contain polynomial inequalities. We call such an extension *semi-algebraic transition system* (SATS). It is easy to see that ATS is a special case of SATS.

Definition 3. *A semi-algebraic transition system is a quintuple $\langle V, L, T, \ell_0, \Theta\rangle$, where V is a set of program variables, L is a set of locations, and T is a set of transitions. Each transition $\tau \in T$ is a quadruple $\langle \ell_1, \ell_2, \rho_\tau, \theta_\tau\rangle$, where ℓ_1 and ℓ_2 are the pre- and post- locations of the transition, $\rho_\tau \in CPF(V, V')$ is the transition relation, and $\theta_\tau \in CPF(V)$ is the guard of the transition. Only if θ_τ holds, the*

transition can take place. Here, we use V' (variables with prime) to denote the next-state variables. The location ℓ_0 is the initial location, and $\Theta \in CPF(V)$ is the initial condition.

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

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

A transition is called *separable* if its relation is a conjunctive formula of equations which define variables in V' equal to polynomial expressions over variables in V. It is easy to see that the composition of two separable transitions is equivalent to a single separable one. An SATS is called separable if each transition of the system is separable. In a separable system, the composition of transitions along a path of the system is also equivalent to a single separable transition. We will only concentrate on separable SATSs as any polynomial program can easily be represented by a separable SATS (see [12]. Any SATS in the rest of the paper is always assumed separable.

For convenience, by $l_1 \overset{\rho_\tau,\theta_\tau}{\rightarrow} l_2$ we denote the transition $\tau = (l_1, l_2, \rho_\tau, \theta_\tau)$, 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 called *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 we assume the first transition with the guard $x + 7 = x^5$, and the second with the guard $x^4 = x + 3$, then the composition of the two guards is $x + 7 = x^5 \wedge (x^4 + 3)^4 = (x^4 + 3) + 3$. That is,

Theorem 2. *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 equivalent to the transition $l_0 \overset{\tau_1;\tau_2;\cdots;\tau_n}{\rightarrow} l_n$.*

Example 5. Consider the SATS $P \mathrel{\widehat{=}} \{V = \{x\}, L = \{l_0, l_1\}, T = \{\tau_1 = \langle l_0, l_1, x' = x^2 + 7, x = 5\rangle, \tau_2 = \langle l_1, l_0, x' = x^3 + 12, x = 12\rangle\}, l_0, \Theta = x = 5\}$. According to the definition, P is separable and $l_0 \overset{\tau_1}{\rightarrow} l_1 \overset{\tau_2}{\rightarrow} l_0$ is a composable transition circle, which is equivalent to $\langle l_0, l_0, x' = (x^2 + 7)^3 + 12, x = 5 \wedge x^2 + 7 = 12\rangle$.

Definition 4 (Ranking Function). *Assume $P = \langle V, L, T, l_0, \Theta\rangle$ is an SATS. A ranking function is a function $\gamma : Val(V) \to \mathbb{R}^+$ such that the following conditions are satisfied:*

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

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

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

where V, V' *denote the starting and ending states of the transition circle, respectively.*

Condition 1 says that for any initial state satisfying the initial condition, its image under the ranking function must be no less than 0; Condition 2 expresses the fact that the value of the ranking function decreases at least c as the program moves back to the initial location along any transition circle, and is still greater than or equal to 0.

According to Definition 4, for any SATS, if we can find such a ranking function, the system will not go through l_0 infinitely often.

4 Discovering Non-linear Ranking Function

In Definition 4, if γ is a polynomial, we call it a *polynomial ranking function*. In this section, we show how to synthesize polynomial ranking functions of an SATS with the techniques for solving SASs.

We will not present a rigorous proof of the approach to synthesize polynomial ranking function, but use the following program as a running example to demonstrate this. We believe, readers can follow the demonstration to derive a proof by their own if interested in.

Example 6. Consider a program shown in Fig.1 (a).

```
    x = m,  m > 0
l0 : while x ≠ 0 do
        if x > 0 then
            x := m1 − x
        else
            x := −x − m2
        end if
    end while
    where m, m1, m2 are integers.
```

(a)

$P = \{$
$\quad V = \{x\}$
$\quad L = \{l_0\}$
$\quad T = \{\tau_1, \tau_2\}$
$\quad \Theta = \{x = m,\ m > 0\}$
$\quad$ where
$\quad\quad \tau_1 : \langle l_0, l_0, x' + x - m_1 = 0, x \geq 1\rangle$
$\quad\quad \tau_2 : \langle l_0, l_0, x' + x + m_2 = 0, x \leq -1\rangle\}$
$\quad \}$

(b)

Fig. 1.

We can transform the program to an SATS as in Fig.1 (b).

Step 1. Predetermine a template of ranking functions. For example, we can assume a template of ranking functions of P in Example 6 in the form $\gamma(\{x\}) = ax + b$, where a, b are parameters.

Step 2– Encoding 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 semi-algebraic system according to Definition 1. Therefore, applying the tool DISCOVERER, we get a necessary and sufficient condition of the derived SAS having no real solution. The condition may contain the occurrences of some program variables. In this case, the condition should hold for any instantiations of these variables. Thus, by introducing universal quantifications of these variables (we usually add a scope to each of these variables according to different situations) and then applying QEPCAD, we can get a necessary and sufficient condition only on the presumed parameters.

Example 7. In Example 6, $\Theta \models \gamma(\{x\}) \geq 0$ is equivalent to that the following parametric SAS has no real solution

$$x = m, m > 0, \gamma(\{x\}) < 0. \tag{3}$$

By calling

$$\textbf{tofind}([x - m], [\,], [-\gamma(\{x\}), m], [\,], [x], [m, a, b], 0),$$

we get that (3) has no real solution iff

$$b + ma \geq 0. \tag{4}$$

By using QEPCAD to eliminate $\forall m > 0$ over (4), we get

$$a \geq 0 \wedge b \geq 0. \tag{5}$$

Step 3–Encoding Decreasing Condition. From Definition 4, there exists a positive constant C such that for any transition circle $l_0 \xrightarrow{\tau_1} l_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_n} l_0$,

$$\rho_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \theta_{\tau_1;\tau_2;\cdots;\tau_n} \models \gamma(V) - \gamma(V') \geq C \wedge \gamma(V') \geq 0. \tag{6}$$

(6) is equivalent to

$$\rho_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \theta_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \gamma(V') < 0 \qquad \text{and} \tag{7}$$

$$\rho_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \theta_{\tau_1;\tau_2;\cdots;\tau_n} \wedge \gamma(V) - \gamma(V') < C \tag{8}$$

both have no real solution. It is easy to see that (7) and (8) are parametric SASs according to Definition 1, so applying the tool DISCOVERER, we obtain some conditions on the parameters. Subsequently, similar to Step 2, we may need to exploit the quantifier elimination tool QEPCAD to simplify the resulted condition in order to get a necessary and sufficient condition only on the presumed parameters.

Example 8. In Example 6, suppose $C > 0$ such that

– For the transition circle $l_0 \xrightarrow{\tau_1} l_0$,
  – firstly, $\rho_{\tau_1} \wedge \theta_{\tau_1} \models \gamma(\{x'\}) \geq 0$ iff

$$x' + x - m_1 = 0, x - 1 \geq 0, \gamma(\{x'\}) < 0 \tag{9}$$

has no solution. Calling

$$\textbf{tofind}([x' + x - m_1], [x - 1], [-\gamma(\{x'\})], [\,], [x'], [x, m_1, a, b], 0),$$

it follows that (9) has no real solution iff

$$b + am_1 - ax \geq 0. \tag{10}$$

(10) should hold for any $x \geq 1$. Thus, by applying QEPCAD to eliminate the quantifier $\forall x \geq 1$ over (10), we get

$$a \leq 0 \wedge am_1 + b - a \geq 0. \tag{11}$$

  – secondly, $\rho_{\tau_1} \wedge \theta_{\tau_1} \models \gamma(\{x\}) - \gamma(\{x'\}) \geq C$ iff

$$x' + x - m_1 = 0, x - 1 \geq 0, \gamma(\{x\}) - \gamma(\{x'\}) < C \tag{12}$$

has no solution. By calling

$$\textbf{tofind}([x' + x - m_1], [x - 1], [\gamma(\{x'\}) - \gamma(\{x\}) + C, C], [\,], [x'], [x, m_1, a, b, C], 0),$$

it results that (12) has no real solution iff

$$2ax - C - am_1 \geq 0. \tag{13}$$

Also, (13) should hold for any $x \geq 1$. Thus, by applying QEPCAD to eliminate the quantifier $\forall x \geq 1$ over (13), we get

$$a \geq 0 \wedge C + am_1 - 2a \leq 0 \wedge C > 0. \tag{14}$$

– Similarly, by encoding the decreasing condition w.r.t. the transition circle $l_0 \xrightarrow{\tau_2} l_0$, we get a condition

$$a \geq 0 \wedge am_2 - b - a \leq 0 \wedge a \leq 0 \wedge C - am_2 + 2a \leq 0. \tag{15}$$

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. If the condition is still complicated, we can utilize the function of PCAD of DISCOVERER or QEPCAD to prove whether the condition can be satisfied. If yes, the tool can produce instantiations of these parameters. Thus, we can get a specific ranking function of the predetermined form by replacing the parameters with the instantiations, respectively.

Example 9. Obviously, for any positive constant C, $C > 0$ always contradicts to (14) and (15). This means that there is no linear ranking functions for the program P.

Note that the above procedure is *complete* in the sense that for any given template of ranking function, the procedure can always give you an answer: yes or no, while an incomplete one (such as the one proposed in [6]) is lack of the ability to produce a negative conclusion.

Example 10. Now, let us consider nonlinear ranking functions of the program in Example 6 in the form $\gamma = ax^2 + bx + c$. For simplicity, we assume $C = 1$.

Applying the the above procedure, we get the condition on m_1, m_2, a, b, c as

$$\begin{aligned} a \geq 0 \wedge c \geq 0 \wedge (b \geq 0 \vee 4ac - b^2 \geq 0) \wedge am_1^2 + bm_1 - 2am_1 + c - b + a \geq 0 \wedge \\ (4ac - b^2 \geq 0 \vee 2am_1 + b - 2a \leq 0) \wedge am_1^2 + bm_1 - 2am_1 - 2b + 1 \leq 0 \wedge \\ am_2^2 - bm_2 - 2am_2 + c + b + a \geq 0 \wedge (4ac - b^2 \geq 0 \vee 2am_2 - b - 2a \leq 0) \wedge \\ am_1 + b \geq 0 \wedge am_2 - b \geq 0 \wedge am_2^2 - bm_2 - 2am_2 + 2b + 1 \leq 0. \quad (16) \end{aligned}$$

For (16), according to m_1 and m_2, we can discuss as follows:

- Let $m_1 = m_2 = 1$, there are quadratic ranking functions for the program, for example $\gamma = x^2$ or $\gamma = 4x^2 + x + 2$;
- Let $m_1 = 1, m_2 = 2$, it is clear that (16) does not hold from its last conjunct. Therefore, the program has no quadratic ranking functions. In fact, the program does not terminate e.g. initializing $m = 2$.
- In the case where $m = 3n \vee m = 3n + 1$ for any positive integer n, the program terminates, and we can also compute a quadratic ranking function $4x^2 - x + 2$ for it.

In the following, we show how to apply this approach to finding ranking functions of a non-linear program.

Example 11. Find a polynomial ranking function for the polynomial program given in Fig.2 (a).

```
    Real x = A
       where 1 < A < 10
l0 : while x > 1 ∧ x < 10 do
       if x > 1 ∧ x < 3 then
          x := x(5 − x)
       else
          x := x + 1
       end if
    end while
```

(a)

$P = \{$
$V = \{x\}$
$L = \{l_0\}$
$T = \{\tau_1, \tau_2\}$
$\Theta = x = A \wedge A > 1 \wedge A < 10$
where
$\tau_1 : \langle l_0, l_0, x' - 5x + x^2 = 0, x > 1 \wedge x < 3 \rangle$
$\tau_2 : \langle l_0, l_0, x' - x - 1 = 0, x \geq 3 \wedge x < 10 \rangle$
$\}$

(b)

Fig. 2.

In order to find a ranking function of the program, we first transform the program to an SATS represented in Fig.2 (b). Then, assume a ranking function template with degree 1 in the form $\gamma(\{x\}) = ax + b$.

After encoding the initial condition and then applying DISCOVERER and QEPCAD, we get a condition on a and b is

$$b + 10\,a \geq 0 \wedge b + a \geq 0. \tag{17}$$

Afterwards, encoding the decreasing condition w.r.t. the transition circle $l_0 \xrightarrow{\tau_1} l_0$ and then applying DISCOVERER and QEPCAD, we obtain

$$b + 4\,a \geq 0 \wedge 4\,b + 25\,a \geq 0 \wedge C + 4\,a \leq 0 \wedge C + 3\,a \leq 0. \tag{18}$$

Similarly, encoding the decreasing condition w.r.t. the transition circle $l_0 \xrightarrow{\tau_2} l_0$ and then applying DISCOVERER and QEPCAD, we get a condition

$$b + 11\,a \geq 0 \wedge b + 4\,a \geq 0 \wedge C + a \leq 0. \tag{19}$$

Thus, a necessary and sufficient condition on these parameters is obtained as

$$C > 0 \wedge a + C \leq 0 \wedge b + 11\,a \geq 0.$$

So, if we assume $C = 1$, we can get a linear ranking function $11 - x$.

For this example, if we assume a ranking function template with degree 2 in the form $\gamma(\{x\}) = ax^2 + bx + c$, and let $C = 1$, we get a necessary and sufficient condition on a, b, c as

$$\begin{aligned} c + 10\,b + 100\,a \geq 0 \wedge c + b + a \geq 0 \wedge b + 9\,a + 1 \leq 0 \wedge b + 21\,a + 1 \leq 0 \wedge \\ (b + 2\,a \geq 0 \vee b + 20\,a \leq 0 \vee 4\,ac - b^2 \geq 0) \wedge 16\,c + 100\,b + 625\,a \geq 0 \wedge \\ c + 4\,b + 16\,a \geq 0 \wedge (b + 8\,a \geq 0 \vee 2\,b + 25\,a \leq 0 \vee 4\,ac - b^2 \geq 0) \wedge \\ 3\,b + 15\,a + 1 \leq 0 \wedge c + 11\,b + 121\,a \geq 0 \wedge c + 4\,b + 16\,a \geq 0 \wedge \\ (b + 8\,a \geq 0 \vee b + 22\,a \leq 0 \vee 4\,ac - b^2 \geq 0) \wedge b + 7\,a + 1 \leq 0. \end{aligned} \tag{20}$$

For (20), applying PCAD in DISCOVERER we get a sample point $(1, -22, 150)$, we therefore obtain a non-linear ranking function $x^2 - 22x + 150$.

5 Conclusions and Future Work

This paper uses the techniques on solving semi-algebraic systems to discover non-linear ranking functions of polynomial programs. This paper also shows how to use computer algebra tools, DISCOVERER and QEPCAD, to synthesize ranking functions for two interesting programs.

The paper represents a part of the authors' efforts to use DISCOVERER to verify programs. We have used it to verify reachability of linear hybrid systems and generate symbolic termination conditions for linear programs in [27], and to discover ranking functions for polynomial programs here. Similar to Cousot's approach [6], DISCOVERER can also be applied to invariant generation for polynomial programs. We will report this in another paper.

Comparing with the well-known tools REDLOG and QEPCAD, DISCOVERER has distinct features on solving problems related to root classification and isolation of SASs through the complete discrimination system. We will analyze its complexity in another paper. The results of the efforts to apply DISCOVERER to program verification are encouraging so far, and we will continue our efforts. The successful story of TERMINATOR [3] also encourages us to develop a program verification tool based on DISCOVERER when we have sufficient experience.

References

1. Braverman, M.: Termination of integer linear programs. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 372–385. Springer, Heidelberg (2006)
2. Bradley, A., Manna, Z., Sipma, H.: Terminaition of polynomial programs. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 113–129. Springer, Heidelberg (2005)
3. Cook, B., Podelski, A., Rybalchenko, A.: TERMINATOR: Beyond safety. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 415–418. Springer, Heidelberg (2006)
4. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. J. of Symbolic Computation 12, 299–328 (1991)
5. Colón, M., Sipma, H.B.: Synthesis of linear ranking functions. In: Margaria, T., Yi, W. (eds.) ETAPS 2001 and TACAS 2001. LNCS, vol. 2031, pp. 67–81. Springer, Heidelberg (2001)
6. Cousot, P.: Proving program invariance and termination by parametric abstraction, Langrangian Relaxation and semidefinite programming. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 1–24. Springer, Heidelberg (2005)
7. Dams, D., Gerth, R., Grumberg, O.: A heuristic for the automatic generation of ranking functions. In: Workshop on Advances in Verification (WAVe'00), pp. 1–8 (2000)
8. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall, Englewood Cliffs (1976)
9. Dolzman, A., Sturm, T.: REDLOG: Computer algebra meets computer logic. ACM SIGSAM Bulletin 31(2), 2–9
10. Floyd, R.W.: Assigning meanings to programs. In: Proc. Symphosia in Applied Mathematics, vol. 19, pp. 19–37 (1967)
11. Hoare, C.A.R.: An axiomatic basis for computer programming. Comm. ACM 12(10), 576–580 (1969)
12. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer, Heidelberg (1995)
13. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 239–251. Springer, Heidelberg (2004)
14. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Non-linear loop invariant generation using Gröbner bases. In: ACM POPL'04, pp. 318–329 (2004)
15. Tarski, A.: A Decision for Elementary Algebra and Geometry. University of California Press, Berkeley (1951)
16. Tiwari, A.: Termination of linear programs. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 70–82. Springer, Heidelberg (2004)

17. International Conference on Verified Software: Theories, Tools and Experiments, ETH Zürich (October 10-13, 2005)
18. Wu, W.-T.: Basic principles of mechanical theorem proving in elementary geometries. J. Syst. Sci. Math. 4, 207–235 (1984)
19. Xia, B., Xiao, R., Yang, L.: Solving parametric semi-algebraic systems. In: Pae, S.-i., Park, H. (eds.) ASCM 2005. Proc. the 7th Asian Symposium on Computer Mathematics, Seoul, December 8-10, pp. 8–10 (2005)
20. Xia, B., Yang, L.: An algorithm for isolating the real solutions of semi-algebraic systems. J. Symbolic Computation 34, 461–477 (2002)
21. Xia, B., Zhang, T.: Real Solution Isolation Using Interval Arithmetic. Comput. Math. Appl. 52, 853–860 (2006)
22. Yang, L.: Recent advances on determining the number of real roots of parametric polynomials. J. Symbolic Computation 28, 225–242 (1999)
23. Yang, L., Hou, X., Xia, B.: A complete algorithm for automated discovering of a class of inequality-type theorems. Sci. in China (Ser. F) 44, 33–49 (2001)
24. Yang, L., Hou, X., Zeng, Z.: A complete discrimination system for polynomials. Science in China (Ser. E) 39, 628–646 (1996)
25. Yang, L., Xia, B.: Automated Deduction in Real Geometry. In: Chen, F., Wang, D. (eds.) Geometric Computation, pp. 248–298. World Scientific, Singapore (2004)
26. Yang, L., Xia, B.: Real solution classifications of a class of parametric semi-algebraic systems. In: Proc. of Int'l Conf. on Algorithmic Algebra and Logic, pp. 281–289 (2005)
27. Yang, L., Zhan, N., Xia, B., Zhou, C.: Program verification by using DISCOVERER. In: Proc. VSTTE'05, Zürich (October 10-October 13, 2005) (to appear)