Program Verification by Using DISCOVERER *

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

[1] Software Engineering Institute, East China Normal University
[2] Laboratory of Computer Science, Institute of Software, Academia Sinica
[3] LMAM & School of Mathematical Sciences, Peking University

Abstract. Recent advances in program verification indicate that various verification problems can be reduced to semi-algebraic system (SAS for short) solving. An SAS consists of polynomial equations and polynomial inequalities. Algorithms for quantifier elimination of real closed fields are the general method for those problems. But the general method usually has low efficiency for specific problems. To overcome the bottleneck of program verification with a symbolic approach, one has to combine special techniques with the general method. Based on the work of complete discrimination systems of polynomials [33, 31], we invented new theories and algorithms [32, 30, 35] for SAS solving and partly implemented them as a real symbolic computation tool in Maple named DISCOVERER. In this paper, we first summarize the results that we have done so far both on SAS-solving and program verification with DISCOVERER, and then discuss the future work in this direction, including SAS-solving itself, termination analysis and invariant generation of programs, and reachability computation of hybrid systems etc.

Key words: *semi-algebraic systems, DISCOVERER, program verification, termination, invariant generation, reachability computation*

1 Introduction

In the last decades, since the first modern computer was invented, the performances of computer hardware have been multiplied by 10^{13} or even more. This is a technical revolution. The immediate consequence is that the size of programs executed on these computers has grown in similar proportions. On the other hand, neither the intellectual capacities of programmers nor the sizes of design and maintenance teams can grow in similar proportions. This results in most software products containing many bugs. For example, we are often told that Microsoft Company releases bug repairing programs for its merchandized products on the website of the company. Some of these errors may cause catastrophic consequences which are very costly and sometimes inadmissible (e.g., nuclear control systems). The difficulty to prevent and find errors grows faster than the

* This work is supported in part by NKBRPC-2002cb312200, NKBRPC-2004CB318003, NSFC-60273022, NSFC-60493200, NSFC-60421001, NSFC-60573007, and NKBRPC-2005CB321902.

size of programs which can now be really huge. Classical software verification methods like code reviews, simulations, tests and so on do not scale up. The production of reliable software, their maintenance and their evolution over long periods of time has become a fundamental concern to computer scientists.

Exploiting mathematical methods to strictly prove that a computer program does exactly what is stated in the program specification has been recognized as an effective and efficient approach to produce reliable softwares and in fact has been made lots of achievements, for example applying model-checking techniques [4–6, 24] to the design of hardware. Since the total program verification problem is undecidable, it is doomed to be impossible to find an universal approach to mechanically verify the correctness of programs without any simplification or restriction. This implies that the possible solution to program verification problem is either by abstract interpretation to simplify the given proof obligation, or by interactive manner to acquire some oracles during verification, or by developing specific methods for specific verification problems which are decidable. Following the above three lines, there are various techniques to program verification that have been well established so far, e.g., the abstraction-based techniques [11] such as static program analysis [12, 16] and program typing [13, 19], theorem-proving based deductive methods [20, 21], model-checking [4–6, 24], etc. The main disadvantage of the abstraction-based techniques is that complicated properties cannot be dealt with well because complicated properties closely interwind with the actual executions of a program, but normally in abstract executions of the program lots of useful information will be lost. The shortcomings of theorem-proving based deductive methods lie in the following three aspects. First, it is semi-automatic and we cannot get anything if theorem provers fail for the given problem. Second, it is not easy to master theorem provers because they are driven by unformalized heuristics, and these heuristics and their interaction are changed often for improving proof strategies. Third, compatibility with other formal methods is somewhat difficult. By contrast, designing a specific full automatic computer-aided method for some specific problems will be more effective and efficient.

Recent advances in program verification indicate that various verification problems of programs, for instance, termination analysis of programs [28, 3], reachability computation of linear hybrid systems [18], and invariant generation [9, 25, 26], can be reduced to SAS solving. Lots of well-known real symbolic computation tools such as REDLOG [15] and QEPCAD [8] have therefore been applied to program verification [18, 9].

Most of the well-known real symbolic computation tools for solving SASs are fundamentally based on the techniques concerning quantifier elimination of real closed fields using the cylindrical algebraic decomposition (CAD) method due to Collins [7], and thus the complexity of the algorithms adopted in these tools is at least double exponential in the number of variables [14]. In order to improve the efficiency of such tools, special methods may be combined with the CAD for specific targets.

Based on the work in [36, 33, 31], two kinds of specific problems of SASs are studied in [32, 30, 35]. Reference [30] presented an algorithm to isolate the real solutions of constant SASs, while references [32, 35] proposed practical algorithms for real solution classification of parametric SASs. We have partly implemented the theory and algorithms as a Maple package named DISCOVERER which has been successfully used to address many problems studied by other researchers [32, 30, 35, 29]. Specifically, when DISCOVERER is applied to program verification, we found that many verification problems can be more effectively solved. In what follows, we will report some of the results, including termination condition generation of linear loop programs and reachability computation of linear hybrid systems.

2 DISCOVERER

In this section, we will give a brief description on DISCOVERER and refer the reader to [32, 35] for details.

All the polynomials discussed in this section are in $\mathbb{Q}[u_1, ..., u_d, x_1, ..., x_s]$, the ring of polynomials in indeterminates $u_1, ..., u_d, x_1, ..., x_s$ with rational coefficients, where $s > 0, d \geq 0$, $u = (u_1, ..., u_d)$ are parameters and $x = (x_1, ..., x_s)$ variables. The following system

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

is called a *semi-algebraic system* and is denoted shortly by

$$[[P], [G_1], [G_2], [H]] \tag{1}$$

where P, G_1, G_2 and H represent the sets of polynomials $\{p_1(u,x), ..., p_s(u,x)\}$ $(= 0)$, $\{g_1(u,x), ..., g_r(u,x)\}$ (≥ 0), $\{g_{r+1}(u,x), ..., g_t(u,x)\}$ (> 0) and $\{h_1(u,x),$ $..., h_m(u,x)\}$ $(\neq 0)$, respectively. An SAS is called *parametric* if $d > 0$ and *constant* otherwise.

Based on the theory of *complete discrimination systems for polynomials* [33, 31], which is a set of explicit expressions in terms of the coefficients of a given polynomial that are sufficient for determining the numbers and multiplicities of the real and imaginary roots of the polynomial, and the *RSD* algorithm [36], which enables us to transform equations into triangular form with good properties, we proposed new algorithms for solving SASs which have been implemented partly in DISCOVERER. Our algorithms and DISCOVERER have been applied to many problems such as automated theorem discovering and proving involving inequalities, computer vision, stability analysis etc., e.g., see [32, 35, 29].

For a constant SAS, T, of the form (1), DISCOVERER can determine the number of distinct real solutions of T, say n, and compute n disjoint cubes with rational vertices. Each of the cubes contains one and only one solution to T and

the width of the cubes can be less than any given positive real. The two functions are realized through calling

$$\texttt{nearsolve}([P],[G_1],[G_2],[H],[x_1,...,x_s])$$

and

$$\texttt{realzeros}([P],[G_1],[G_2],[H],[x_1,...,x_s],w),$$

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

For a parametric SAS, T, of the form (1) and any integer $N \geq 0$, DISCOVERER can determine the conditions on u such that T has exactly N distinct real solutions for x. The conditions can be obtained by combining two functions of DISCOVERER: `tofind` and `Tofind`. First, by calling

$$\texttt{tofind}([P],[G_1],[G_2],[H],[x_1,...,x_s],[u_1,...,u_d],N),$$

one obtains the *border polynomial* BP in u of the system T and the necessary and sufficient conditions for T to have N distinct real solutions provided BP$\neq 0$. Second, to determine the situation when parameters are on the "boundary", i.e., BP$= 0$, one need to call `Tofind`. Suppose R is a factor of BP, one can call

$$\texttt{Tofind}([P,R],[G_1],[G_2],[H],[x_1,...,x_s],[u_1,...,u_d],N)$$

to obtain conditions when the parameters are on $R = 0$.

The last argument of `tofind` and `Tofind` is one of the following three forms:

- a non-negative integer b, to indicate that T has exactly b distinct real solutions;
- a range $b..c$, where b, c are non-negative integers and $b < c$, to indicate that T has b to c distinct real solutions;
- a range $b..w$, where b is a non-negative integer and w a name, to indicate T has more than or equal to b distinct real solutions.

3 Termination Analysis and Reachability Computation with DISCOVERER

In this section, we summarize the work that we have achieved so far on termination condition generation of linear programs and reachability computation of linear hybrid systems with DISCOVERER.

3.1 Termination Criterion Generation of Linear Programs

The well-known method for establishing termination of programs is the use of well-founded domains together with so-called ranking functions that map the state space of a program to a well-founded domain. Clearly, the existence of

such a ranking function implies termination. Some heuristics on generating linear ranking functions for linear programs have been proposed, e.g. [10, 23].

Generally speaking, the cost for synthesizing linear ranking functions is very high, for example, the complexity of the algorithm in [10] is exponential and that of the algorithm in [23] could be double exponential. Besides, it is not easy to characterize programs to which such an algorithm can be applied. If the algorithm fails for a given program, we can conclude nothing about the termination of the program. The ideal solution to termination problem is to establish a criterion for each class of programs whose termination problem is decidable so that for a given program contained in the class, it can be decided if the program terminates trivially by computing the criterion, although the cost for generating such a criterion is also expensive.

Reference [28] made an attempt towards the direction by showing the decidability of a class of programs of the form, called linear loop programs,

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

by relating the termination of $\mathtt{P}_1$ to the positive eigenvalues of A, where A and B are $N \times N$ and $N \times M$ matrices respectively, $Bx > b$ represents a conjunction of linear inequalities in the program variables and $x := Ax + c$ represents the linear assignments to each of the variables. In order to establish criteria for termination of $\mathtt{P}_1$ based on Tiwari's work, theories and tools concerning root classification of parametric SASs on an interval are demanded.

Based on Tiwari's work, we used DISCOVERER and established termination conditions for programs of the following form:

$$\mathtt{P}_{11}:\ \mathtt{while}\ (\textstyle\sum_{i=1}^{n} c_i x_i > 0)\ \{x := Ax\}.$$

For example, suppose the dimension of x is specified, say, three, setting $A = (a_{ij})_{3\times 3}$, we have:

Theorem 1. *Provided the polynomial system* $\{(a_{11}-\lambda)x_1+a_{12}x_2+a_{13}x_3,\ a_{21}x_1+(a_{22}-\lambda)x_2+a_{23}x_3,\ a_{31}x_1+a_{32}x_2+(a_{33}-\lambda)x_3,\ c_1x_1+c_2x_2+c_3x_3\}$ *has no non-trivial solutions, the program* $\mathtt{P}_{11}$ *terminates if and only if the following formula is satisfied:*

$$(p \geq 0 \wedge q \geq 0 \wedge r \geq 0) \vee (D_3 < 0 \wedge r \geq 0), \tag{2}$$

where

$$\begin{aligned} p &= -a_{11} - a_{22} - a_{33}, \\ q &= -a_{21}a_{12} + a_{11}a_{33} - a_{31}a_{13} + a_{22}a_{33} + a_{11}a_{22} - a_{23}a_{32}, \\ r &= -a_{11}a_{22}a_{33} + a_{31}a_{13}a_{22} + a_{21}a_{12}a_{33} + a_{11}a_{23}a_{32} \\ &\quad -a_{31}a_{12}a_{23} - a_{21}a_{13}a_{32}, \\ D_3 &= -4\,q^3 + 18\,p\,q\,r + p^2q^2 - 4\,p^3r - 27\,r^2. \end{aligned}$$

One can use various algebraic tools, such as multivariate resultant methods, to check whether a certain polynomial system has a non-trivial solution or not.

4 Reachability Computation of Linear Hybrid Systems

Reachability and safety are now being recognized as central problems in designing hybrid and dynamical systems. Contrast with issues of stability and controllability of hybrid systems which are well-studied in control theory, there are not many results on reachability of those systems in computer science. Known classes of hybrid systems for which the reachability problem is decidable are timed automata [2], multirate automata [1], rectangular hybrid automata [22, 17], etc. Recently a most general decidability result of three families of linear hybrid systems whose differential equations of the form $\dot{\xi} = A\xi + Bu$ was obtained in [18], where $\xi(t) \in \mathbb{R}^n$ is the state of the system at time t, $A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{n\times m}$ are the system matrices, and $u : \mathbb{R} \to \mathbb{R}^m$ is a piecewise continuous function which is called the control input.

Roughly speaking, computing a reach set in [18] is via the following two steps: firstly, purely mathematically transform a problem of reachability computation to an SAS and therefore the decidability is obtained according to the famous Tarski's results [27]; then use the well-known computer algebra programs REDLOG and QEPCAD to solve the resulting SAS.

Using DISCOVERER instead of REDLOG and QEPCAD in the second step, we found that the results reported in [18] can be much improved. This can be justified by revisiting the examples in [18] that are not dealt well with REDLOG and QEPCAD. We list two of them below.

Example 1 (Example 3.5 of [18]). Consider the linear control vector field given by the diagonal matrix $A \in \mathbb{Q}^{2\times 2}$ and $\mathcal{U} = \{u\}$ defined as follows:

$$A = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix}, \quad u(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} -ae^{\frac{1}{2}t} \\ ae^t \end{bmatrix}, \text{ with } a \geq 0.$$

Let the initial set be $X = \{(0, 0)\}$. Then, after mathematical transformation, we get the reach set from X forwards as:

$$\text{Post}(X) = \{(y_1, y_2) \in \mathbb{R}^2 \mid \exists a \exists z : 0 \leq a \wedge z \geq 1 \wedge h_1 = 0 \wedge h_2 = 0\} \tag{3}$$

where

$$h_1 = y_1 - \frac{2}{3}a(-z^4 + z),\ h_2 = y_2 z^2 - \frac{1}{2}a(z^4 - 1).$$

Since the quantifiers in (3) cannot be eliminated using REDLOG or QEPCAD alone, reference [18] applied REDLOG to eliminate a first and then using QEPCAD to eliminate z, and thus obtained $\text{Post}(X)$ is

$$\{(y_1, y_2) \in \mathbb{R}^2 \mid (y_2 > 0 \wedge\ y_1 + y_2 \leq 0) \vee (y_2 < 0 \wedge\ y_1 + y_2 \geq 0) \\ \vee 4y_2 + 3y_1 = 0\} \tag{4}$$

With DISCOVERER, (3) can be solved by calling first

`tofind`$([h_1, h_2], [a, z - 1], [\,], [\,], [z, a], [y_2, y_1], 1..n)$;

and we get that the system has real solutions if and only if

$$y_2 > 0 \wedge y_1 + y_2 < 0$$

provided that

$$y_1 \neq 0, y_2 \neq 0, y_1 + y_2 \neq 0 \text{ and } R \neq 0$$

where

$$R = 192y_2^3y_1^2 - 63y_1^3y_2^2 + 112y_1y_2^4 - 6y_1^4y_2 + 3y_1^5 + 16y_2^5.$$

Further considering (y_1, y_2) on the "boundaries" with DISCOVERER (by calling `Tofind`), we get that the system has real solutions if and only if

$$\begin{aligned}&(y_2 > 0 \wedge\ y_1 + y_2 < 0) \vee (y_1 = y_2 = 0) \vee \\ &(y_1 < 0 \wedge (\ y_2 \text{ is the smallest root of } R = 0 \text{ when } \ y_1 \text{ is specified})).\end{aligned} \tag{5}$$

The whole computation costs no more than 30 seconds on a PC (Pentium III/800 GHz CPU, 256M main memory, Windows XP) with Maple 9.

Note that (5) is inconsistent with (4). We give the following counter-examples by DISCOVERER to show that (4) is not correct. Suppose $y_2 = -1, y_1 = 2$, thus $y_2 < 0 \wedge y_1 + y_2 \geq 0$. It is easy to see that the system has no real solutions under $a \geq 0$ and $z \geq 1$. Another two counter-examples are $y_2 = 1, y_1 = -1$ and $y_1 = 4, y_2 = -3$.

Example 2 (Example 3.7. of [18]). Consider the control linear vector field given by the matrix $A \in \mathbb{Q}^{2\times 2}$ and $\mathcal{U}$ defined as follows:

$$A = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \quad u(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} a\cos(2t) \\ -a^{-1}\sin(2t) \end{bmatrix}, \text{ with } a > 0.$$

Let the initial set be $X = \{(0, 0)\}$ and the final set be $Y = \{(-1, 1)\}$. Then, after some mathematical transformation, Y is reachable from X if and only if the following formula holds:

$$\exists w \exists z \exists a : a > 0 \wedge g_1 = 0 \wedge g_2 = 0 \wedge g_3 = 0, \tag{6}$$

where $g_1 = w^2 + z^2 - 1$, $g_2 = w((4a^2 - 2)z + 2 - a^2) + 3a$ and $g_3 = (a^2 - 2)(w^2 - z^2 + z) - 3a$.

The solution to (6) in [18] with REDLOG and QEPCAD is as follows: Eliminate w with REDLOG first and then obtain

$$\begin{aligned}\exists z \exists a :\ & a > 0 \\ & \wedge 16a^6z^4 - 24a^6z^3 + 9a^6z^2 - a^6z + 48a^5z^2 - 24a^5z + 3a^5 - 48a^4z^4 \\ & +84a^4z^3 - 42a^4z^2 + 6a^4z - 9a^4 - 48a^3z^2 + 60a^3z - 12a^3 + 36a^2z^4 \\ & -84a^2z^3 + 60a^2z^2 - 12a^2z + 18a^2 + 12az^2 - 24az + 12a - 8z^4 \\ & +24z^3 - 24z^2 + 8z = 0 \wedge \\ & 16a^4z^4 - 8a^4z^3 - 15a^4z^2 + 8a^4z - a^4 - 16a^2z^4 + 20a^2z^3 + 12a^2z^2 \\ & -20a^2z + 13a^2 + 4z^4 - 8z^3 + 8z - 4 = 0\end{aligned} \tag{7}$$

with the assumption that $4a^2z - a^2 - 2z + 2 \neq 0$. The resulting formula is so complex that the quantifiers cannot be automatically eliminated using either REDLOG or QEPCAD. Thus $z = 0$ is assumed and it results that (6) holds provided that

$$\begin{aligned} &\exists a: \ a > 0 \wedge (a^2 \neq 2) \wedge \\ &\qquad 3a^4 - 9a^3 - 12a^2 + 18a + 12 = 0 \wedge -a^4 + 13a^2 - 4 = 0 \end{aligned} \tag{8}$$

which is found to be true using QEPCAD. Alternatively, reference [18] also used REDUCE to derive the roots of the polynomials in a of (8) and found the common root $r = \frac{1}{2}(\sqrt{17} + 3) \approx 3.56156$ satisfying $r > 0$ and $r^2 \neq 2$. Hence, $y = (-1, 1)$ is reachable from $x = (0, 0)$ by taking $a = r$.

In contrast, we solve (6) using DISCOVERER simply by executing the following command

$$\texttt{realzeros}([g_1, g_2, g_3], [\,], [a], [\,], [w, z, a], 1/1000000)$$

and get the following three solutions within 0.3 seconds on the same machine

$$\begin{array}{l} [\,[\frac{-3786927439}{4294967296}, \frac{-30295419511}{34359738368}], [\frac{16599516199973}{35184372088832}, \frac{33199032472081}{70368744177664}], [\frac{938055701}{268435456}, \frac{938055737}{268435456}]\,], \\ [\quad [\frac{-2797709061}{4294967296}, \frac{-699427265}{1073741824}], \quad [\frac{-6517535813}{8589934592}, \frac{-6517535651}{8589934592}], \quad [\frac{70273639}{134217728}, \frac{281094637}{536870912}]\,], \\ [\qquad\qquad [1, 1], \qquad\qquad [0, 0], \qquad\qquad [\frac{14938235}{4194304}, \frac{29876471}{8388608}] \quad], \end{array}$$

which approximates to

$$\begin{array}{l} [\,[-.8817127531,-.8817127531], \quad [.4717866261,.4717866271], \quad [3.494529802,3.494529936]\,], \\ [\,[-.6513924014,-.6513924012], \, [-.7587410292,-.7587410103], \, [.5235794112,.5235795621]\,], \\ [\qquad\qquad [1.,1.], \qquad\qquad [0.,0.], \qquad\qquad [3.561552763,3.561552882]\,]. \end{array}$$

It is easy to see that the solution given in [18] with REDLOG,QEPCAD and REDUCE is just one of the three solutions we got above using DISCOVERER.

5 Future Work

In this paper, it has been shown that the theory for isolating real solutions of a constant SAS and classifying real solutions of a parametric SAS and the tool DISCOVERER invented in [32, 30, 35] can indeed improve program verification, e.g., termination condition generation of linear programs and reachability computation of linear hybrid systems. But all what we have done up to now is just a first step, as there are still many challenging problems left in this field from the theory on SASs itself to program verification.

As for future work on program verification, we think that it is worth investigating the following issues: Firstly, as termination condition generation for the simple linear loop programs with DISCOVERER, it deserves to apply this method to more general linear programs or even non-linear programs. In addition, it is also a challenging problem to study reachability for more complicated

hybrid systems with DISCOVERER. Finally, we believe that it is possible to produce interesting results by applying the theory and DISCOVERER to invariant generation of programs. The dominant method on finding invariants of a program is by abstract interpretation [11], but this method suffers from the problem of overshooting invariants. With comparison to the abstract interpretation based techniques, references [9, 25, 26] proposed a new technique for invariant generation by reducing this problem to SAS solving so that the new approach can avoid weak invariant generation. However, the high complexity of the well-known techniques and tools for non-linear constraint solving limits the applicability of this approach, for example, in order to avoid producing non-linear constraints [25, 26] had to sacrifice the completeness of the method. Additionally, using the approach of [25, 26] all the resulting invariants can only consist of polynomial equations but reject polynomial inequalities, this may thus restrict the expressiveness of resulting invariants.

Regarding future work on DISCOVERER itself, all theories and algorithms implemented in DISCOVERER only concern SASs with 0-dimensional solutions (the number of solutions is finite) until now. But, in fact, many problems in program verification may be reduced to SASs with positive dimensional solutions (the number of solutions is infinite). This demands us to extend the established theories and tool in order that SASs with positive dimensional solutions can be also well addressed. Furthermore, we shall develop special tools based on DISCOVERER for program verification.

Finally, we have to point out that up to now we have no idea on how to generalize our approach to non-numerical problems like determining termination for term-rewriting systems. It seems difficult (impossible) because the set of symbols of a term-rewriting system that play as coefficient in SASs does not satisfy required algebraic properties such as the axioms for field.

Acknowledgement

The authors sincerely thank the anonymous referees for their useful comments which are helpful for improving the presentation of this paper.

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. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183-235, 1994.
3. A.R. Bradley, Z. Manna, H.B. Sipma. Termination of Polynomial Programs. In Proc. of Verification, Model Checking and Abstract Interpretation (VMCAI'05), LNCS 3385, 2005.
4. E.M. Clark, A. Emerson and A.P. Sistla. Automatic verification of finite-state concurrent programs using temporal logic. *ACM Transaction on Programming Languages and Systems,* 8(2):244-263, 1986.

5. 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, 1981.6.
6. E.M. Clarke, O. Grumberg and D.A. Peled. *Model Checking.* MIT Press, 1999.
7. G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. *Automata Theory and Formal Languages,* LNCS 33, pp. 134–183, 1975.
8. G. E., Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. of Symbolic Computation*, 12:299–328, 1991.
9. 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.
10. M. colón and H.B. Sipma. Synthesis of linear ranking functions. In TACAS'01, LNCS 2031, pp. 67–81, 2001.
11. 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.
12. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In ACM POPL'79, pp. 269-282, 1979.
13. L. Damas and R. Milner. Principal type-schemes for functional programs. In ACM POPL'82, pp.207-212, 1982.
14. J. H., Davenport and J. Heintz. Real Elimination is Doubly Exponential. *J. of Symbolic Computation,* 5:29–37, 1988.
15. A. Dolzman and T. Sturm. REDLOG: Computer algebra meets computer logic. *ACM SIGSAM Bulletin,* 31(2):2–9.
16. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretation complete. *J. ACM,* 4792:361-416,2000.
17. T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *J. of Computer Science and System Sciences,* 57:94-124, 1998.
18. 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.
19. R. Milner. A theory of polymorphism in programming. *J. Computer System Science,* 17(3):348-375, 1978.
20. S. Owre, J.M. Rushby and N. Shankar. PVS: A protype verification system. In CADE'92, LNCS 607, pp. 748-752, 1992.
21. C. Paulin-Mohring and B. Werner. Synthesis of ML programs in the system Coq. *J. Symbolic Logic,* 15(5/6):607-640, 1993.
22. A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusions. In CAV'94, LNCS 818, pp. 95-104, Springer-Verlag, 1994.
23. A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In VMCAI'04, LNCS 2937, pp. 239–251, 2004.
24. J.-P. Queille and J. Sifakis. Verification of concurrent systems in CESAR. In Int. Symp. On Programming, LNCS 137, pp.337-351, 1982.
25. S. Sankaranarayanan, H.B. Sipma, and Z. Manna. Non-linear loop invariant generation using Gröbner bases. In ACM POPL'04, pp. 318–329, 2004.
26. S. Sankaranarayanan, H.B. Sipma, and Z. Manna. Constructing invariants for hybrid systems. In HSCC'04, LNCS 2993, pp. 539–554.
27. A. Tarski. *A Decision for Elementary Algebra and Geometry.* University of California Press, Berkeley, May. 1951.
28. A. Tiwari. Termination of linear programs. In CAV'04, LNCS 3114, pp. 70–82, 2004.

29. D. Wang and B. Xia. Stability analysis of biological systems with real solution classification. In: *Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation* (ISSAC 2005) (M. Kauers, ed.), pp. 354–361. ACM Press, New York (2005).
30. B. Xia and L. Yang. An algorithm for isolating the real solutions of semi-algebraic systems. *J. Symbolic Computation,* 34:461–477, 2002.
31. L. Yang. Recent advances on determining the number of real roots of parametric polynomials. *J. Symbolic Computation*, 28:225–242, 1999.
32. 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).
33. L. Yang, X. Hou and Z. Zeng. A complete discrimination system for polynomials. *Science in China (Ser. E)*, 39:628–646, 1996.
34. L. Yang and B.C. Xia, An explicit criterion to determine the number of roots of a polynomial on an interval. *Progress in Natural Science*, 10(12):897–910, 2000.
35. L. Yang and B. Xia. Real solution classifications of a class of parametric semi-algebraic systems. In *Proc. of Int'l Conf. on Algorithmic Algebra and Logic*, pp. 281–289, 2005.
36. L. Yang, J. Zhang and X. Hou. A criterion of dependency between algebraic equations and its applications. In *Proceedings of International Workshop on Mathematics Mechanization 1992*, Wu, Wen-tsun & Cheng, Min-de (eds.). International Academic Publishers, Beijing, pp. 110–134, 1992.