A Way of Validating Polynomial Program with Semi-algebraic System

Qinggeng Jin[1, a], Lijuan Su[1, b], Anping He[1, c, *] and Juxia Xiong[1, 2, d]

[1]Guangxi Key Laboratory of Hybrid Computer and IC Design Analysis, Nanning, China;

[2]Chengdu Institute of Computer Application, Chinese Academy of Science, Chengdu, China

[a]jinqinggeng@aliyun.com, [b]1151593068@qq.com, [c, *]dr.he@sohu.com, [d]287047039@qq.com

Keywords: Polynomial program, Semi-algebraic System, Symbolic model checking.

Abstract. Program checking is an interesting and challenging problem. Many kinds of programs can be formulated as polynomial programs, expressing the same meanings in a mathematical way. Moreover, with the mathematic framework of semi-algebraic systems, the polynomial based model checking can be applied directly and efficiently, as well as shows a good solution to reduce the state explosion problem. In this article, we show an easy way to translate the polynomial program into a semi-algebraic transition system, and then check the properties by computing the zeros.

1 Introduction

The design of reliable software is a grand challenge in computer science in the 21st century, as our modern life becomes more and more computerized [1]. One of the bases for designing reliable software is the correctness of programs [1], the needs from applications and reality of the researches make program verification both interesting and challenging.

In this paper, we show the model checking procedure of polynomial program. Many kinds of programs can be formulated as polynomial programs, which expresses the same meanings in a mathematical way. Instead of only focusing on theorem proving, e.g. invariant generation in [2], our method is based on "algebraic" symbolic model checking [3, 4]. This article shows the semi-algebraic systems ($SASs$) can be applied to symbolic model checking of polynomial programs. The main idea is that the polynomial program is expressed as semi-algebraic transition systems ($SATSs$) directly, the properties are translated into polynomials, and then the checking problem is concluded by finding the common zeros of the semi-algebraic systems restricted by the polynomials [3].

The rest of this paper is organized as follows: Section 2 reviews the basic notations of symbolic model checking, semi-algebraic system and related knowledge are listed in section 3, section 4 introduces polynomial programs and their formal representations, says transition system. Then in section 5, we shall propose a method for checking the polynomial programs, and we end in section 6 with some ideas for future work.

2 Symbolic Model Checking

Model checking is a method for formally verifying finite-state concurrent systems. Specifications about the system are expressed as temporal logic formulae, and efficient algorithms are used to traverse the model defined by the system and check if the specification holds or not. The model checking algorithm, which is based on the manipulation of boolean formulae, is called symbolic model checking. The system model is represented by *labelled transition systems*, which are usually called *kripke structures*.

Definition 2.1 (Kripke Structure) [13] Kripke structure is a tuple , $K = \langle S, S_0, R, AP, L \rangle$, where

— S is a finite set state;

— S_0 is the set of initial states;

— $R \subseteq S \times S$ is a transition relation;

— AP is a set of all atomic propositions and their negative propositions;

— $L : S \to 2^{AP}$ is the labelling function.

Properties are specified by temporal logic, here we use the *Computation Tree Logic* (CTL), which is a subset of modal branching time logic defined by Clarke and Emerson [5]. In CTL, temporal operators consist of A or E; followed by G (Global), F (Future), X (neXt), or U (Until). The syntax of CTL formula is given as follows:

1. Every atomic proposition is a CTL formula.
2. if f and g are CTL formulae, then so are

$$\neg f, f \wedge g, AX\ f, EX\ f, A(f\, U\ g), E(f\, U\ g)$$

The other operators can be derived from these according to the following rules:

$$(1)\ f \vee g = \neg(\neg f \wedge \neg g);\ (2)\ AF\ g = A(true\ U\ g);\ (3)\ EF\ g = E(true\ U\ g);$$

$$(4)\ AG\ f = \neg E(true\ U\ \neg f);\ (5)\ EG\ f = \neg A(true\ U\ \neg f).$$

Symbolic CTL model checking is to check $(\forall s \in S_0[K, s \models \mathrm{p}])$. The notation $K, s \models \mathrm{p}$ means that the CTL formula p is true in state s of kripke model K.

Tarski verified that in lattice a monotonic function f has a least fixpoint, denoted $\mu x.f$ and s greatest fixpoint, denoted $\upsilon x.f$.

Theorem 2.1 For any Kripke structure $K = (S, S_0, R, AP, L)$, there are

1. $EF\ p = \mu y.(p \vee EX\ y)$; 2. $EG\ p = \upsilon y.(p \wedge EX\ y)$; 3. $E(p\ U\ q) = \mu y.(q \wedge EX\ y)$.

The other CTL formulae can be derived from the above formulae. Then the CTL operators can be characterize in terms of fixpoints of appropriate functions [3]. In the end, we need only to calculate the formula $EX\ \phi$ to a formula ϕ for fixpoint obtained by computing $EX\ y$ iteratively.

3 Semi-algebraic system

In this section, we review the theories of semi-algebraic system[9].

Let $\mathrm{K}[x_1, \cdots, x_n]$ be the ring of polynomials in n indeterminates, $\mathbf{X}=\{x_1, \cdots, x_n\}$, with coefficients in the field K. Let the variables be ordered as $x_1 \prec x_2 \prec \cdots \prec x_n$. Then, the *leading variable* (or *main variable*) of a polynomial p is the variable with the biggest index which indeed occurs in p [11]. 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 $c_i s$ are polynomials in $\mathrm{K}[x_1, \cdots, x_{k-1}]$. We call $c_m x_k^m$ the leading term of p w.r.t. x_k and c_m the *leading coefficient*. An *atomic polynomial* formula over $\mathrm{K}[x_1, \cdots, x_n]$ is of the form $p(x_1, \cdots, x_n) \triangleright 0$, where $\triangleright \in \{=, >, \geq, \neq\}$, while a polynomial formula over $\mathrm{K}[x_1, \cdots, x_n]$ is constructed from atomic polynomial formulae by applying the logical connectives [11]. We will denote by $PF(\{x_1, \cdots, x_n\})$ the set of polynomial formulae and by $CPF(\{x_1, \cdots, x_n\})$ the set of conjunctive polynomial formulae, respectively. Moreover, we will use □ to stand for rationales and □ for reals, and fix K to be □ [10].

In the following, the n indeterminates are divided into two groups: u= $(u_1, \cdots, u_t)$ and x= $(x_1, \cdots, x_n)$, which are called parameters and variables, respectively, and we sometimes use “,” to denote the conjunction of atomic formulae for simplicity [10].

Definition 3.1 [10] (Semi-algebraic system) A *semi-algebraic* system is a conjunctive polynomial formula of following from:

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

where $r > 1$, $l \geq k \geq 0$, $m \geq 0$ and all $p_i's$, $g_i's$ and $h_i's$ are in $\mathbb{Q}[u,x]\setminus\mathbb{Q}$. A SAS of the form (1) is called parametric if $t = 0$, otherwise constant [10].

4 Polynomial program and transition system

A *polynomial program* comprises a finite set *Proc of procedure* names with one distinguished procedure *Main* [6], execution of which starts with a call to *Main*. But in this paper, we only introduce the flat program, that is to say, polynomial program has been synthesized firstly, as well as only one procedure will be considered (like in figure 1).

The n-tuple X=$(x_1,\cdots,x_n)$ consists of the variables appearing in the polynomial program, one or many programs counter variables ("pc") range over the program labels, and some common key words, e.g. "integer", "if", "then" and "while", appear with their standard meanings like which in other programs. Now, let us introduce the rules of labelling polynomial programs. Let P be a polynomial program, and the labelled version is denoted by P^{ς}. Then the rules of labelling the polynomial are like which in [5]:

— If P is not a composite statement (like $x := e$), then $P^{\varsigma} = P$.

— If $P = P_1; P_2$, then $P^{\varsigma} = P_1^{\varsigma}; l; P_2^{\varsigma}$.

— If $P =$ **if** b **then** P_1 **else** P_2 **end if**, then $P^{\varsigma} =$ **if** b **then** $l_1 : P_1^{\varsigma}$ **else** $l_2 : P_2^{\varsigma}$ **end if**.

— if $P =$ **while** b **do** P_1 **end while**, then $P^{\varsigma} =$ **while** b **do** $l_1 : P_1^{\varsigma}$ **end while**.

— If $P =$ **begin** $P_1 \| P_2 \| \cdots \| P_n$ **end**, then $P^{\varsigma} =$ **begin** $l_1 : P_1 l_1' \| l_2 : P_2 l_2' \| \cdots \| l_n : P_n l_n'$ **end**.

4.1 Semi-algebraic transition system

Instead of representing polynomial program by *control flow graph* (like in [6, 7]), we formalize this kind of programs with *transition systems*, which facilitates the checking procedure better.

Firstly, we rewrite the polynomial programs with *command formulae* [8]. Let us show this with the program in figure 1. The command formula c of above polynomial program from line 3 to 4 over variables x is

$$c \equiv \underbrace{pc = 3 \wedge x > 0}_{guard} \wedge \underbrace{x' = 1 - x \wedge pc' = 4}_{action}$$

In a command formula, the subformula over unprimed variables $x_1,\cdots,x_n$ forms the *guard* (enabling condition). The remaining conjuncts form the *action* (update of the variables). Usually, they are of the form $x' = E$, where E is the update expression over unprimed variables (translating assignments $x := E$).

So the polynomial program could be given as a set C of command formulae.The translation from programs to sets of command formulae is standard for this kind of programming languages.

Now we can get the semi-algebraic transition system (SATS) [1, 2] by command formulae directly. SATS is extended the notion of algebraic transition systems (ATS) [14], in which each transition is equipped with a conjunctive polynomial formula as guard, and contains both polynomial equations and inequations.

Definition 4.1 (Semi-algebraic transition system) A semi-algebraic transition system is a quintuple $\langle V, L, T, l, \Theta \rangle$, where V is a set of program variables, L is a set of locations, and L is a set of transitions [9]. Each transition $\tau \in T$ is a quadruple $\langle l_1, l_2, \rho_\tau, \theta_\tau \rangle$, where l_1 and l_2 are the pre- and postlocations 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 [9]. Only if θ_τ holds, the transition can take place. Here, V' (variables with prime) denotes the next-state variables. The location l_0 is the initial location, and $\Theta \in CPF(V)$ is the initial condition [9].

For $SATS$, a state is an evaluation of the variables in V and all states could be denoted by $Val(V)$ [9]. Without confusions 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 [9]. Then the polynomial program described in figure 1 could be translated into

$$P = \{V = \{x\} \quad L = \{l_0\} \quad T = \{\tau_1, \tau_2\}$$

$$\text{Where } \tau_1 : \langle l_0, l_0, x' + x - 1 = 0, x \geq 1 \rangle$$

$$\tau_2 : \langle l_0, l_0, x' + x + 2 = 0, x \leq -1 \rangle$$

$$\Theta = \{x = 100\}\}$$

For convenience, by $l_1 \xrightarrow{\rho_\tau, \theta_\tau} l_2$ we denote the transition $\tau = (l_1, l_2, \rho_\tau, \theta_\tau)$, or simply by $l_1 \xrightarrow{\tau} l_2$ [9]. A sequence of transitions $l_{11} \xrightarrow{\tau_1} l_{12}, \cdots, l_{n1} \xrightarrow{\tau_n} l_{n2}$ is called *compassable* if $l_{i2} = l_{(i+1)1}$ for $i = 1, \cdots, n-1$, and written as $l_{11} \xrightarrow{\tau_1} l_{12}(l_{21}) \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_n} l_{n2}$ [9]. A compassable sequence is called *transition circle* at l_{11}, if $l_{11} = l_{n2}$. For any compassable sequence $l_0 \xrightarrow{\tau_1} l_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_n} l_n$, it is easy to show that there is a transition of the form $l_0 \xrightarrow{\tau_1;\tau_2;\cdots;\tau_n} l_n$ so that the compassable sequence is equivalent to the transition [1], 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 $\tau_1; \tau_2; \cdots; \tau_n, \rho_{\tau_1}; \rho_{\tau_2}; \cdots; \rho_{\tau_n}$ and $\theta_{\tau_1}; \theta_{\tau_2}; \cdots; \theta_{\tau_n}$, 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 [12]. 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 $x + 7 = x^5 \wedge (x^4 + 3)^4 = (x^4 + 3) + 3$ [12]. That is to say, for any compassable sequence $l_1 \xrightarrow{\tau_1} l_2, \cdots, \xrightarrow{\tau_n} l_n$, it is equivalent to the transition $l_0 \xrightarrow{\tau_1;\tau_2;\cdots;\tau_n} l_n$ [1]. Moreover, according to the definition in [4, 5], the transitions in $SATS$ are *compassable*.

Let us consider the $SATS$ $P \equiv \{V = \{x\}, L = \{l_0, l_1\}, T = \{\tau_1 = \langle l_0, l_1, x' = x^2 + 7, x = 5 \rangle, \tau_2 = \langle l_1, l_0, x' = x^3 + 12, x = 12 \rangle\}, l_0, \Theta \equiv x = 5\}$ [1]. $l_0 \xrightarrow{\tau_1} l_1 \xrightarrow{\tau_2} l_0$ is a compassable transition circle, which is equivalent to $\langle l_0, l_0, x' = (x^2 + 7)^3 + 12, x = 5 \wedge x^2 + 7 = 12 \rangle$ [1].

It is also convenient to add a set of polynomial to ρ_τ to modify or restrict a transition τ, sometimes. We can simply denote this by $\tau \cap P$ where P is a polynomial set. Moreover, we also simply denote $S \cap P$ for all transitions in $SATS$ S restricted by set P.

5 Checking polynomial program

In this section, we propose a method to process the model checking polynomial programs.

5.1 Polynomial semantics of CTL

The first step to translate CTL formulae into polynomials is how to deal with the quantification operators. Let ϕ be a CTL formula and the corresponding polynomial be $[\![\phi]\!]$, then

Definition 5.1 (Polynomial semantics of quantification operators) Let ϕ be a CTL formula and $x_1, \cdots, x_n$ be the variables involved in ϕ in the field κ, the quantification operators will be:

$$\llbracket \exists x_i f\left(x_1,\cdots,x_n\right) \rrbracket = \bigvee_{l\in\kappa} \llbracket f\left(x_1,\cdots,x_n\right)\big|_{x_i\leftarrow l} \rrbracket$$

$$\llbracket \forall x_i f\left(x_1,\cdots,x_n\right) \rrbracket = \bigwedge_{l\in\kappa} \llbracket f\left(x_1,\cdots,x_n\right)\big|_{x_i\leftarrow l} \rrbracket$$

Then it is convenient to translate CTL formulae into their corresponding polynomial representations.

Definition 5.2 (Polynomial semantics of CTL formulae) Let Kripke structure, the polynomial semantics will be:

（1）$\llbracket x \rrbracket, x \in AP$ （2）$\llbracket \neg\phi \rrbracket = \neg_p \llbracket \phi \rrbracket$ （3）$\llbracket \phi \wedge \psi \rrbracket = \llbracket \phi \rrbracket \wedge_p \llbracket \psi \rrbracket$

（4）$\llbracket \phi \vee \psi \rrbracket = \llbracket \phi \rrbracket \vee_p \llbracket \psi \rrbracket$ （5）$\llbracket EX\phi(\bar{x}) \rrbracket = \llbracket \exists \bar{x}' \left(R(\bar{x},\bar{x}') \wedge \phi(\bar{x}')\right) \rrbracket$

（6）$\llbracket AX\phi(\bar{x}) \rrbracket = \neg_p \llbracket EX\neg\phi \rrbracket$ （7）$\llbracket E(\phi \cup \psi) \rrbracket = \llbracket \phi \rrbracket \vee_p \left(\llbracket \phi \rrbracket \wedge_p \llbracket EX\ E(\phi \cup \psi) \rrbracket \right)$

（8）$\llbracket A(\phi \cup \psi) \rrbracket = \llbracket \phi \rrbracket \vee_p \left(\llbracket \phi \rrbracket \wedge_p \llbracket AX\ A(\phi \cup \psi) \rrbracket \wedge_p \llbracket EX\ A(\phi \cup \psi) \rrbracket \right)$

```
integer  $x = 100$;
while ($x \neq 0$)  do
    $l_0 : if\,(x > 0)$  then
        $l_1 : x := 1 - x$;
    else
        $l_1 : x := -x - 2$;
    end if
end while
```

Figure 1 A program to multiply two numbers

```
procedure fixpoint
input
    polynomial program  $P$
    LTL formula  $\phi$
    a map  $f : x_i \rightarrow x_i^{'} (1 \le i \le n)$
begin
    translate  $P$ into SATS  $S = \langle V, L, T, l, \Theta \rangle$
    translate  $\phi$ into polynomial set  $P$
    $S^{'} = S \cap P$
    computing the zero of  $S^{'}$
end
```

Figure 2 Computing the fixpoint

```
procedure Check($P, R$)
input
polynomial program  $P$
    LTL formula  $\phi$
    a map  $f : x_i \rightarrow x_i^{'} (1 \le i \le n)$
begin
    Case
    $P \in AP$; Return  $P$;
    $P = \neg\phi$; Return  $\neg_p Check(\phi, R)$;
    $P = \phi \wedge \psi$; Return  $Check(\phi, R) \wedge Check(\psi, R)$;
    $P = \phi \vee \psi$; Return  $Check(\phi, R) \vee Check(\psi, R)$;
    $P = EX\phi$; Return  $EX\,(Check(\phi, R))$;
    $P = AX\phi$; Return  $\neg_p EX\,(Check(\neg\phi, R))$;
    $P = E(\phi \cup \psi)$ ; Return  $Quntil(E, Check(\phi, R), Check(\psi, R))$;
    $P = A(\phi \cup \psi)$; Return  $Quntil(A, Check(\phi, R), Check(\psi, R))$;
end
```

Figure 3 Model checking algorithm

5.2 Algorithms

We have shown in the figure 2, the most complex and basic step for checking these programs is how to calculate the fixpoint, let us see the lemma below:

Lemma 5.1 Given a $SATS\ S$ and a set of polynomials P corresponding to the formula ϕ, then there is a map $f : x_i \mapsto x_i' \left(1 \le i \le n\right)$ so that formula $EX\ \phi$ represented by $S \cap f\left(P\right)$.

Theorem shows that a fixpoint can be got by computing $EX\ y$ iteratively. Hence, a property can be translated into polynomials directly, and we use theories of semi-algebraic systems to calculate zeros of polynomials, which represent the fixpoint. The algorithm is shown in figure 3.

Our model checking algorithms are the same as ones in [3, 4], which do polynomial based model checking by computing the fixpoints of polynomials set other than $SATS$. We do not list the details here to bother readers.

6 Conclusion

The polynomial program is a natural expression of some real system, especially while considering a system in terms of its performance. In this article, we propose a procedure of validating the polynomial program in a formal way. The basic idea is to convert the polynomial program into a semi-algebraic transition system, and then check the properties by the way of computing its zeros iteratively. In future, we would connect the tools listed in this article to make the validating procedure smoothly.

Acknowledgements

This work was financially supported by the National Natural Science Foundation of China under Grants nos. (11371003,11461006 and 61402121), the Natural Science Foundation of Guangxi under Grants nos. (2013GXNSFAA019342 and 2012GXNSFGA060003), the ”Bagui Scholar” project of Guangxi, the Scientific and technological research projects in Guangxi Universities under Grants nos. (ZD2014044), the Scientific Research Fund of Guangxi Education Department under Grants nos. (201012MS274), the graduate education innovation project in 2015 of Guangxi University for Nationalities under Grants nos. (gxun-chxs2015097).

References

[1] Yinghua Chen, Bican Xia, Lu Yang, Naijun Zhan, Chaochen Zhou, Discovering Non-linear Ranking Functions by Solving Semi-algebraic Systems. ICTAC 2007: 34-49.

[2] Yinghua Chen, Bican Xia, Lu Yang, Naijun Zhan, Generating Polynomial Invariants with DISCOVERER and QEPCAD. Formal Methods and Hybrid Real-Time Systems 2007: 67-82.

[3] Weibo Mao, Jinzhao Wu, Application of Wu's method to symbolic model checking. ISSAC 2005: 237-244.

[4] George S. Avrunin, Symbolic model checking using algebraic geometry. In CAV, Proceedings of the 8th International Conference on Computer Aided Verification, pages 26-27.

[5] Edmund M. Clarke, Jr., Orna Grumberg and Doron A. Peled, Model Checking, MIT Press, 1999, ISBN 0-262-03270-8.

[6] Markus Möuller-Olm, Michael Petter, Helmut Seidl, Interprocedurally Analyzing Polynomial Identities. STACS 2006: 50-67.

[7] M. Möuller-Olm, H. Seidl, Computing polynomial program invariants, Information Processing Letters 91 (5) (2004) 233-244.

[8] Andreas Podelski, Andrey Rybalchenko, Software Model Checking of Liveness Properties via Transition Invariants, MPI Technical Report 2-004, 2003.

[9] Xiaoyan Zhao, Non-termination Analysis of Polynomial Programs by Solving Semi-Algebraic Systems. Advances in Multimedia, Software Engineering and Computing Vol.1 2012: 205-211.

[10] Bican Xia, DISCOVERER: A tool for solving semi-algebraic systems, http://www.is.pku.edu.cn/ xbc/software.html.

[11] Bin Wu, Xiao Guang Zou, Computing Invariants for Hybrid Systems. Applied Mechanics and Materials 2013: 556-561.

[12] Cliff B. Jones, Zhiming Liu, Jim Woodcock, Theoretical Aspects of Computing － ICTAC 2007. Lecture Notes in Computer Science, ISBN: 978-3-540-75290-5(Print) 978-3-540-75292-9 (Online).

[13] Randal E. Bryant, Pankaj Chauhan, Edmund M. Clarke, Amit Goel, A Theory of Consistency for Modular Synchronous Systems. Lecture Notes in Computer Science, 2002: 523-541.

[14] Sankaranarayan, S. Sipma, H. B. Manna, Non-linear Loop Invariant Generation using Grobner Bases ACM symposium on principles of programming language. 2004, VOL 31, pages 318-329.