A Way of Validating Polynomial Program with Semi-algebraic System

Qinggeng Jin\textsuperscript{1,a}, Lijuan Su\textsuperscript{1,b}, Anping He\textsuperscript{1,c,*} and Juxia Xiong\textsuperscript{1,2,d}

\textsuperscript{1}Guangxi Key Laboratory of Hybrid Computer and IC Design Analysis, Nanning, China;
\textsuperscript{2}Chengdu Institute of Computer Application, Chinese Academy of Science, Chengdu, China

\textsuperscript{a}jqinggeng@aliyun.com, \textsuperscript{b}1151593068@qq.com, \textsuperscript{c}dr.he@sohu.com, \textsuperscript{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 (SATSSs) 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.

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

\begin{itemize}
  \item \( S \) is a finite set state;
  \item \( S_0 \) is the set of initial states;
  \item \( R \subseteq S \times S \) is a transition relation;
  \item \( AP \) is a set of all atomic propositions and their negative propositions;
  \item \( L : S \to 2^{AP} \) is the labelling function.
\end{itemize}
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 $\neg f, f \land g, AX f, EF f, A(f \land g), E(f \land g)$

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

1. $f \lor g = \neg \neg f \land \neg g$; (2) $AF g = A(true \land g)$; (3) $EF g = E(true \land g)$;
2. $AG f = \neg E(true \land \neg f)$; (5) $EG f = \neg A(true \land \neg f)$.

Symbolic CTL model checking is to check $\forall s \in S[K,s \models p]$. The notation $K, s \models p$ 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 greatest fixpoint, denoted $\nu x.f$.

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

1. $EF p = \mu y.(p \lor EX y)$; 2. $EG p = \nu y.(p \land EX y)$; 3. $E(p \land q) = \mu y.(q \land EX y)$.

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 $\phi$ for fixpoint obtained by computing $EX \phi$ iteratively.

### 3 Semi-algebraic system

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

Let $K[x_1, \ldots, x_n]$ be the ring of polynomials in $n$ indeterminates, $X = \{x_1, \ldots, x_n\}$, with coefficients in the field $K$. Let the variables be ordered as $x_1 < x_2 < \cdots < x_n$. 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 $K[x_1, \ldots, x_{k-1}]$. 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 $K[x_1, \ldots, x_n]$ is of the form $p(x_1, \ldots, x_n) > 0$, where $\in \{=, >, \geq, \neq\}$, while a polynomial formula over $K[x_1, \ldots, x_n]$ is constructed from atomic polynomial formulae by applying the logical connectives [11]. 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. Moreover, we will use $\square$ to stand for rationales and $\Box$ for reals, and fix $K$ to be $\mathbb{F}$ [10].

In the following, the $n$ indeterminates are divided into two groups: $u = (u_1, \ldots, u_l)$ and $x = (x_1, \ldots, x_n)$, 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:

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

(1)
where \( r > 1, \ l \geq k \geq 0, \ m \geq 0 \) and all \( p_i', g_i', h_i' \) are in \( \mathbb{R}[u,x] \). 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 \( P^\omega \). Then the rules of labelling the polynomial are like which in [5]:

- If \( P \) is not a composite statement (like \( x := e \)), then \( P^\omega = P \).
- If \( P = P_1; P_2 \), then \( P^\omega = P_1^\omega; l; P_2^\omega \).
- If \( P = \text{if } b \text{ then } P_1 \text{ else } P_2 \text{ end if} \), then \( P^\omega = \text{if } b \text{ then } P_1^\omega \text{ else } l_2; P_2^\omega \text{ end if} \).
- If \( P = \text{while } b \text{ do } P_1 \text{ end while} \), then \( P^\omega = \text{while } b \text{ do } l_1; P_1^\omega \text{ end while} \).
- If \( P = \text{begin } P_1 \text{ end } \cdots \text{end } P_n \), then \( P^\omega = \text{begin } l_1; P_1^\omega \text{ end } \cdots \text{end } l_n; P_n^\omega \).

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

\[
c = p_c = 3 \land x > 0 \land x = 1 - x \land p_c = 4
\]

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 \( \theta_\tau \) 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 $\text{Val}(V)$ [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

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

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 $l_1 \xrightarrow{\rho} l_2$ we denote the transition $\tau = (l_1, l_2, \rho, \theta)$, or simply by $l_1 \xrightarrow{\tau} l_2$ [9]. A sequence of transitions $l_{i_1} \xrightarrow{\tau_1} l_{i_2} \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_{i_n}} l_{i_2}$ is called compassable if $l_{i_2} = l_{(i_1)}$ for $i = 1, \ldots, n-1$, and written as $l_{i_1} \xrightarrow{\tau_1} l_{i_2} \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_{i_n}} l_{i_2}$ [9]. A compassable sequence is called transition circle at $l_{i_1}$, if $l_{i_1} = l_{i_2}$. 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 $\theta_{\tau_1 \cdots \tau_n}$ are the compositions of $\tau_1, \tau_2, \cdots, \tau_n, \rho_{\tau_1 \tau_2 \cdots \tau_n}$, 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 $x = 7 = x^3$, and the second with the guard $x^4 = x + 3$, then the composition of the two guards is $x + 7 = x^3 \wedge (x^4 + 3)^2 = (x^4 + 3)^3 + 3$ [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 $l_0 \xrightarrow{\tau_1 \cdots \tau_n} l_n$ [1]. Moreover, according to the definition in [4, 5], the transitions in SATS are compassable.

Let us consider the SATS $P = \{V = \{x\}, L = \{l_0\}, T = \{\tau_1 = \{l_0, l_0, x' = x^2 + 7, x = 5\}, \tau_2 = \{l_0, l_0, x' = x^3 + 12, x = 12\}\}, L_0, \Theta = \{ x=5 \}$ [1]. $l_0 \xrightarrow{\tau_1} l_1 \xrightarrow{\tau_2} l_0$ is a compassable transition circle, which is equivalent to $\{l_0, l_0, x' = (x^2 + 7)^3 + 12, x = 5 \wedge x^3 + 7 = 12\}$ [1].

It is also convenient to add a set of polynomial to $\rho_{\tau}$ to modify or restrict a transition $\tau$, 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 $\phi$ be a CTL formula and the corresponding polynomial be $\mathbb{P}\phi$, then

**Definition 5.1 (Polynomial semantics of quantification operators)** Let $\phi$ be a CTL formula and $x_1, \cdots, x_n$ be the variables involved in $\phi$ in the field $\kappa$, the quantification operators will be:
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. \( \exists x \cdot \phi = \bigvee_{i} f(x_1, \ldots, x_n) \big|_{\phi \in E} \)
2. \( \forall x \cdot \phi = \bigwedge_{i} f(x_1, \ldots, x_n) \big|_{\phi \in E} \)

**Figure 1** A program to multiply two numbers

**Figure 2** Computing the fixpoint

**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 $\phi$, then there is a map $f : x_i \mapsto x'_i (1 \leq i \leq n)$ so that formula $EX \phi$ represented by $S \cap f(P)$.

Theorem shows that a fixpoint can be got by computing $EX \phi$ 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


