Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 607013, 12 pages
http://dx.doi.org/10.1155/2015/607013

Research Article

A Deductive Approach towards Reasoning about Algebraic Transition Systems

Jun Fu,[1,2] Jinzhao Wu,[2] and Hongyan Tan[3]

[1] *School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China*
[2] *Guangxi Key Laboratory of Hybrid Computation and IC Design Analysis, Guangxi University for Nationalities, Nanning 530006, China*
[3] *High Performance Network Lab, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China*

Correspondence should be addressed to Jinzhao Wu; himrwujzh@aliyun.com

Received 7 June 2015; Accepted 16 August 2015

Academic Editor: Krishnaiyan Thulasiraman

Algebraic transition systems are extended from labeled transition systems by allowing transitions labeled by algebraic equations for modeling more complex systems in detail. We present a deductive approach for specifying and verifying algebraic transition systems. We modify the standard dynamic logic by introducing algebraic equations into modalities. Algebraic transition systems are embedded in modalities of logic formulas which specify properties of algebraic transition systems. The semantics of modalities and formulas is defined with solutions of algebraic equations. A proof system for this logic is constructed to verify properties of algebraic transition systems. The proof system combines with inference rules decision procedures on the theory of polynomial ideals to reduce a proof-search problem to an algebraic computation problem. The proof system proves to be sound but inherently incomplete. Finally, a typical example illustrates that reasoning about algebraic transition systems with our approach is feasible.

1. Introduction

System verification requires a mathematical structure on which the system in question is described precisely. *Labeled transition systems* [1] are such structures proposed for this purpose, which are widely used to specify hardware and software systems [2], for example, integrated circuit system, communication protocols, and concurrent algorithms. A labeled transition system is a specified transition system (first presented by Keller [3]) whose transitions are labeled by abstract labels. Abstract labels are sufficient for modelling atomic actions which trigger transitions of systems, but they are insufficient to describe enough details on transitions of complex systems. For instance, we concern much with the details on how a train reduces its speed in the brake mode, which is usually specified by mathematical equations.

Algebraic transition systems [4] are extended from labeled transition systems by labeling transitions with algebraic assertions, which are conjunctions of polynomial equations. Transitions labeled with algebraic assertions are able to describe how states change according to those polynomial equations. That is very necessary for modelling complex systems. What is more significant is that many mathematical techniques on polynomials are available to the analysis of complex systems, such as the theory of polynomial ideals [5]. On the other hand, conventional methods are not competent for the verification for algebraic transition systems due to the complexity of algebraic assertions. To the best of our knowledge, there is no approach for reasoning about algebraic transition systems. Our study is motivated mainly by this.

Our approach is related to *theorem proving* which is a well-established verification method of labeled transition systems. The theorem proving method [6, 7] tries to find a proof of the desired property, which is written as a theorem in logic languages. Another verification method, called *model checking*, uses a finite-state traversal technique algorithm [8, 9]. Hence model checking method automatically checks whether a given system satisfies the desired properties by traversing the state space of the system. However, model checking requires systems to be finite-state systems or those systems whose state space can be divided into finite quotient subspaces [10, 11]. While theorem proving method is not restricted by finite-state systems and hence applies to complex systems, most of which have infinite state spaces. Since the state spaces of algebraic transition systems are defined on $\mathbb{R}^n$ which is infinite, we choose theorem proving method to verify of algebraic transition systems.

Inspired by [4, 12–14], we present a deductive approach for specifying and verifying algebraic transition systems. Our approach includes a modification of dynamic logic (*ADL*) and a proof system for *ADL*. The *ADL* is extended from dynamic logic [15] by allowing algebraic equations in modalities. There are two standard modalities $[\alpha]$ and $\langle\alpha\rangle$ where α is defined with algebraic equations. The $[\alpha]$ refers to the states reachable by all runs of α, while $\langle\alpha\rangle$ indicates the states reachable by some runs of α. The formal semantics of modalities is defined with zero sets of polynomials. These modalities embedded in logical formulas are used to model behaviors of algebraic transition systems. The properties of algebraic transition systems are specified with *ADL* formulas. The satisfaction of formulas is defined with zero sets of polynomials and the semantics of modalities. For deciding whether the desired properties are satisfied, a proof system of the sequent-calculus style, called *ADL* calculus, is constructed. This proof system aims to find a proof of the desired properties with inference rules. Several special rules are customized to handle modalities with algebraic equations by reducing the proof-search problem to an algebraic computation problem. The algebraic computation procedures enhance the reasoning power of our proof systems. The proof system is proved to be sound but inherently incomplete as many other proof systems. Reasoning about algebraic transition systems with our approach is demonstrated with a typical example.

In recent decades, the deductive approach for specifying and verifying transition systems has received fruitful results [16]. TLA$^+$ [17] is a specification language designed by Lamport for formally describing and reasoning about distributed systems. Systems are specified in TLA$^+$ as formulas of the Temporal Logic of Actions (TLA) [6], which is a variant of temporal logic. The TLA$^+$ proof system (TLAPS) [18, 19] is a general platform for development of TLA$^+$ proofs. A whole proof in TLAPS is decomposed into a collection of subproofs which are sent to backend verifiers including SMT solvers, theorem provers, and proof assistants. Compared with TLA$^+$ and TLAPS, our approach is designed for the direct proof of properties on algebraic transitions. With our approach the proof problem is reduced to an algebraic computation problem such as the ideal membership problem on ideal theory. Our proof system can be considered as a backend verifier of TLAPS for algebraic transition systems.

Combined with mathematical procedures, the deductive approach can be used for the verification of more complex systems, for example, real-time systems and reactive systems [20–22]. Platzer [12, 23] developed a deductive framework for the verification of hybrid systems, which are dynamic systems containing continuous evolutions and discrete transitions. A discrete transition in [12] is specified as an explicit assignment of a variable. For instance, the primed variable x' in the discrete transition $x' = f(x_1, \ldots, x_n)$, which assigns the value of $f(x_1, \ldots, x_n)$ to x', can be immediately eliminated by a replacement with $f(x_1, \ldots, x_n)$. In contrast, a transition in algebraic transition system is modelled as an algebraic equation. Consider the transition $f(x', x_1, \ldots, x_n) = 0$ as an example. The primed variable x' in this transition will be directly eliminated only if x' can be equivalently written as a polynomial on $x_1, \ldots, x_n$, such as $x' = f'(x_1, \ldots, x_n)$. In most cases, the transitions in algebraic transition systems generalise the discrete parts of hybrid systems. Algebraic transition systems cannot simply be seen as subsets of hybrid systems and therefore are not covered by usual methods. Somehow our approach can be considered as a complement to usual methods for verifying complex systems.

The rest of this paper is organized as follows. Section 2 presents some preliminary concepts and some theorems which lie in the core of our approach. We introduce our understanding of algebraic transition systems in Section 3. The algebraic modification of dynamic logic is described in Section 4. In Section 5 we construct a proof system for this logic and prove the soundness and inherent incompleteness of the proof system in Section 7. Our approach is illustrated by reasoning about a train control system in Section 6. Section 8 concludes with some ideas for future work.

2. Preliminary

In this section, we introduce several important conclusions on polynomial ideal theory, which lie in the core of our approach.

We begin with the concepts of polynomials and ideals. Let $\mathbb{N}$ be the set of natural numbers including 0, $\mathbb{R}$ the set of reals, and $\mathbb{C}$ the set of complex numbers obtained as the *algebraic closure* of the reals. Let $\mathbf{V} = \{x_1, \ldots, x_n\}$ be a set of variables. The set of polynomials on the variables, whose coefficients are drawn from the reals, is denoted by $\mathbb{R}[x_1, \ldots, x_n]$.

Definition 1 (zero set). Let $f(x_1, \ldots, x_n) \in \mathbb{R}[x_1, \ldots, x_n]$ be a polynomial on $\mathbf{V}$; the zero set of $f(x_1, \ldots, x_n)$, denoted by $\mathrm{Zero}(f)$, is the set of points in the complex plane such that

$$\mathrm{Zero}(f) = \{\vec{z} \in \mathbb{C}^n \mid f(\vec{z}) = 0\}, \tag{1}$$

where $f(\vec{z})$ is obtained from $f(x_1, \ldots, x_n)$ by replacing all variables with the elements of the point $\vec{z}$.

We write f instead of $f(x_1, \ldots, x_n)$ when the variables are understood in the context. Let $F = \{f_1, \ldots, f_s\}$ be a finite set of polynomials over $\mathbb{R}[x_1, \ldots, x_n]$; its corresponding zero set is defined as

$$\mathrm{Zero}(F) = \{\vec{z} \in \mathbb{C}^n \mid f_i(\vec{z}) = 0,\ \forall 1 \le i \le s\}. \tag{2}$$

We say the polynomial f *vanishes* at the set $\mathrm{Zero}(F)$ if $f(\vec{z}) = 0$ for all $\vec{z} \in \mathrm{Zero}(F)$.

Definition 2 (ideals). A subset $I \subseteq \mathbb{R}[x_1, \ldots, x_n]$ is an ideal, if and only if

(1) $0 \in I$;

(2) for all $f_1, f_2 \in I$, $(f_1 + f_2) \in I$;

(3) if $f \in I$ and $g \in \mathbb{R}[x_1, \ldots, x_n]$, then $fg \in I$, where fg indicates the product of polynomials f, g.

An ideal generated by a set of polynomials $F = \{f_1, \ldots, f_s\}$, denoted by $\langle F \rangle$, is the smallest ideal containing F and equivalently

$$\langle F \rangle = \left\{ \sum_{i=1}^{s} g_i f_i \mid g_i \in \mathbb{R}[x_1, \ldots, x_n],\ \forall 1 \le i \le s \right\}. \quad (3)$$

The ideal $\langle F \rangle$ is said to be finitely generated if the set F is finite. Hilbert's basis theorem says that every ideal in $\mathbb{C}[x_1, \ldots, x_n]$ is finitely generated.

The basic relation of an ideal and its generators is that they have the same zero set according to the following theorem.

Theorem 3. *Given an ideal $I = \langle F \rangle$ generated by $F = \{f_1, \ldots, f_s\}$, then the zero set of I and the zero set of F are the same:*

$$\text{Zero}(I) = \text{Zero}(F). \quad (4)$$

Proof. (1) Since $F \subset I$, we immediately conclude that $\text{Zero}(F) \supset \text{Zero}(I)$. That is,

$$\begin{aligned} \vec{z} \in \text{Zero}(I) &\Longrightarrow (\forall f \in I)[f(\vec{z}) = 0] \\ &\Longrightarrow (\forall f \in F)[f(\vec{z}) = 0] \Longrightarrow \vec{z} \in \text{Zero}(F). \end{aligned} \quad (5)$$

(2) Conversely,

$$\begin{aligned} \vec{z} \in \text{Zero}(F) &\Longrightarrow f_1(\vec{z}) = \cdots = f_s(\vec{z}) = 0 \\ &\Longrightarrow (\forall h = g_1 f_1 + \cdots + g_s f_s)[h(\vec{z}) = 0] \\ &\Longrightarrow (\forall h \in I)[h(\vec{z}) = 0] \Longrightarrow \vec{z} \in \text{Zero}(I). \end{aligned} \quad (6)$$

□

Definition 4 (radical ideal). Let $I \subset \mathbb{R}[x_1, \ldots, x_n]$ be an ideal. The *radical* of I, denoted by $\sqrt{I}$, is the set

$$\sqrt{I} = \{f \mid (\exists n \in \mathbb{N})[f^n \in I]\}. \quad (7)$$

The following theorem asserts a significant relation between zero sets and ideal membership, which is the underlying algebraic principle of axiom rules in Section 5.

Theorem 5. *Given $f, f_1, \ldots, f_s \in \mathbb{R}[x_1, \ldots, x_n]$ and $F = \{f_1, \ldots, f_s\}$, if there is an integer $q \in \mathbb{N}$ such that $f^q \in \langle F \rangle$, that is, $f \in \sqrt{\langle F \rangle}$, then f vanishes at the zero set of F; that is, $\text{Zero}(F) \subset \text{Zero}(f)$. Equivalently,*

$$f \in \sqrt{\langle F \rangle} \Longrightarrow \text{Zero}(F) \subset \text{Zero}(f). \quad (8)$$

Proof. This theorem immediately corresponds to one direction of the famous theorem called *Hilbert's Nullstellensatz*. The proof of Hilbert's Nullstellensatz can be found in [5]. □

A fundamental question in ideal theory is checking whether a given polynomial belongs to the radical of an ideal, which is known as *radical membership problem*. This problem involves the following theorem.

Theorem 6. *Let $f \in \mathbb{R}[x_1, \ldots, x_n]$ be a polynomial and let $I = \langle f_1, \ldots, f_s \rangle \subseteq \mathbb{R}[x_1, \ldots, x_n]$ be an ideal. Then f belongs to the radical of the ideal I if and only if the constant 1 belongs to the ideal $\tilde{I} = \langle f_1, \ldots, f_s, 1 - yf \rangle \subset \mathbb{R}[x_1, \ldots, x_n, y]$; that is,*

$$f \in \sqrt{I} \Longleftrightarrow 1 \in \langle f_1, \ldots, f_s, 1 - yf \rangle, \quad (9)$$

where y is a new variable different from $x_1, \ldots, x_n$.

Proof. A proof of this theorem can be found in any standard text on ideal theory (see Proposition 8 in [5]). □

The core of solving radical membership problem requires a special kind of generators, called *reduced Gröbner basis*. Every ideal of $\mathbb{R}[x_1, \ldots, x_n]$ has a unique finite reduced Gröbner basis [24]. To determine if $f \in \sqrt{\langle f_1, \ldots, f_s \rangle} \subset \mathbb{R}[x_1, \ldots, x_n]$, we compute the reduced Gröbner basis of the ideal $\langle f_1, \ldots, f_s, 1 - yf \rangle \subset \mathbb{R}[x_1, \ldots, x_n, y]$. If the result is $\{1\}$, then $f \in \sqrt{I}$. Otherwise, $f \notin \sqrt{I}$.

Another application of reduced Gröbner basis, shown by the following theorem, is deciding whether there exists a zero set for a finite set of polynomials.

Theorem 7. *Let $F \subset \mathbb{R}[x_1, \ldots, x_n]$ be a finite set of polynomials and G the reduced Gröbner basis for $\langle F \rangle$. Then F has an empty zero set if and only if $1 \in G$; that is, $\text{Zero}(F) = \emptyset \Leftrightarrow 1 \in G$.*

Proof. A proof of this theorem can be found in Corollary 4.3.7 in text [24]. □

3. Algebraic Transition Systems

In this section, we demonstrate how algebraic assertions enrich the abstract labels of labeled transition systems.

Definition 8 (algebraic assertions). An *algebraic assertion* ψ over the set of variables $\mathbf{V}$ is defined as a finite union of polynomial equations of the form

$$\psi \overset{\text{def}}{=} f_1 = 0 \land f_2 = 0 \land \cdots \land f_s = 0, \quad (10)$$

where, for each $1 \le i \le s$, $f_i \in \mathbb{R}[x_1, \ldots, x_n]$.

For an algebraic assertion ψ, its zero set is defined as

$$\text{Zero}(\psi) = \{\vec{v} \in \mathbb{C}^n \mid f_i(\vec{v}) = 0,\ \forall 1 \le i \le s\}. \quad (11)$$

We say that a point $\vec{v} \in \mathbb{C}^n$ satisfies ψ, denoted by $\vec{v} \models \psi$, if $\vec{v}$ belongs to the zero set of ψ; that is, $\vec{v} \in \text{Zero}(\psi)$.

An algebraic transition system is specialized from a labeled transition system. Each transition of an algebraic transition system is labeled with an algebraic assertion instead of an abstract label.

Definition 9 (algebraic transition system). An algebraic transition system $\mathscr{A}$ is a tuple $\mathscr{A} = \langle \mathscr{S}, \mathscr{T}, \Psi, \lambda \rangle$, where

(i) $\mathscr{S}$ is the set of states;

(ii) $\mathscr{T} \subset \mathscr{S} \times \mathscr{S}$ is the set of transitions;

(iii) Ψ is a set of algebraic assertions on $\mathbf{V} \cup \mathbf{V}'$ including the null label τ;

(iv) $\lambda : \mathscr{T} \rightarrow \Psi$ is a label function assigning each transition to an algebraic assertion.

For an algebraic transition system $\mathscr{A}$, a state $s \in \mathscr{S}$ is a function which maps each variable in $\mathbf{V}$ to a real. According to the label function λ, each transition $t \in \mathscr{T}$ is labeled with an algebraic assertion denoted by $\lambda(t) \in \Psi$. The algebraic assertion $\lambda(t)$ is defined on $\mathbf{V} \cup \mathbf{V}'$, where $\mathbf{V}$ denotes the current-state variables and $\mathbf{V}'$ denotes the next-state variables.

The transition relation of $\mathscr{A}$, which describes how states change, is defined by algebraic assertions on $\mathbf{V} \cup \mathbf{V}'$. For each $t \in \mathscr{T}$, the transition relation ρ_t is determined by the label $\lambda(t)$ as follows:

$$\rho_t = \left\{ \left\langle s, s' \right\rangle \mid s, s' \in \mathscr{S} \wedge \lambda(t)\left(s, s'\right) = 0 \right\}, \qquad (12)$$

where s, s' indicate the current state and the next state, respectively, and $\lambda(t)(s, s')$ is evaluated by substituting each variable $v \in \mathbf{V}$ of $\lambda(t)$ with the corresponding value in s and each variable $v' \in \mathbf{V}'$ with the corresponding value in s', respectively. In particular, the null label τ specifies an identical relation on $\mathscr{S}$; that is, $\rho_\tau = \{\langle s, s\rangle \mid s \in \mathscr{S}\}$. The transition labeled by ψ from s to s' is denoted by $s \xrightarrow{\psi} s'$.

An algebraic transition system is *deterministic* if there is at most one transition and one label for any state; otherwise it is *nondeterministic*. As for a deterministic algebraic transition system, the next state is determined uniquely by the current state. For instance, given an algebraic assertion $\psi \overset{\text{def}}{=} x' - x - 1 = 0$, the next state is obtained by adding 1 to the variable x in the current state. We say the transition labeled by ψ is deterministic and nondeterministic if $\psi \overset{\text{def}}{=} (x')^2 - x^2 - 3x - 2 = 0$ (because the next state can take $x' = x + 1$ or $x' = x + 2$). Obviously, an algebraic transition must be deterministic if for all $\psi \in \Psi$ each variable $x' \in \mathbf{V}'$ in ψ can be written as a unique polynomial over $\mathbf{V}$. In this case, each algebraic assertion can be written as $\psi \overset{\text{def}}{=} \bigwedge_i x_i' = f_i(x_1, \ldots, x_n)$ with each $x_i' \in \mathbf{V}'$ and $x_1, \ldots, x_n \in \mathbf{V}$. Hence the value of each variable x_i' in the next state is uniquely determined by $x_1, \ldots, x_n$ in the current state according to $f_i(x_1, \ldots, x_n)$.

Definition 10 (run). Given an algebraic transition system $\mathscr{A} = \langle \mathscr{S}, \mathscr{T}, \Psi, \lambda \rangle$, a run σ of $\mathscr{A}$ is defined by a sequence of transitions as follows:

$$\sigma : s_0 \xrightarrow{\psi_0} s_1 \xrightarrow{\psi_1} s_2 \cdots, \qquad (13)$$

where the ith element of σ is denoted by σ_i and for each $i \geq 0$ there exists a transition $\sigma_i = \langle s_i, s_{i+1} \rangle \in \mathscr{T}$ from state $s_i \in \mathscr{S}$ to state $s_{i+1} \in \mathscr{S}$ such that

$$\begin{gathered} \lambda\left(\sigma_i\right) = \psi_i, \\ \psi_i\left(s_i, s_{i+1}\right) = 0. \end{gathered} \qquad (14)$$

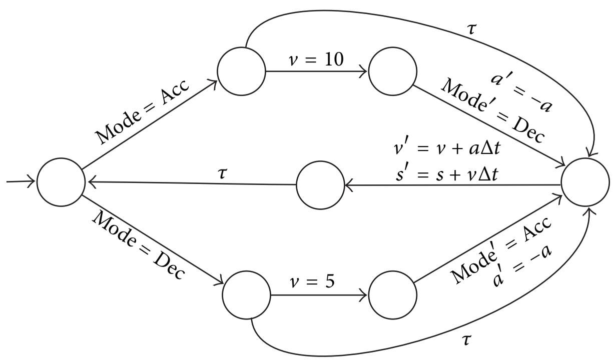

Figure 1: A simple train control system.

Example 11. In order to illustrate algebraic transition systems, we present a simplified train control system shown in Figure 1. Assume that a train has two modes: the acceleration mode (Acc) and the deceleration mode (Dec). The train keeps checking the current mode and velocity. If it is in mode Acc and its velocity reaches 10, it will invert the acceleration power ($a = -a$) and change its mode to mode Dec. Then the position s of the train evolves with velocity v along $s = s + v\Delta t$ and $v = v + a\Delta t$. If the velocity of the train slows down to 5 in mode Dec, it will invert its deceleration power ($a = -a$) and switch to mode Acc. Compared with real-time systems and hybrid system, the behavior of algebraic transition systems is discrete, such as the discrete behavior of the train with time period Δt. Note that we use the relaxed version of algebraic assertions. For instance, we write $a' = -a$ as the relaxed version of $a' + a = 0$. The Dec and Acc can be any certain constants.

In contrast with classical labeled transition systems with abstract labels [3], algebraic transitions systems are widely useful for modeling data flows, due to algebraic assertions describing how data changes between states in detail. What is more significant is that the introduction of concepts on ideal theory leads to the presence of more powerful and efficient algebraic methods for reasoning about complex systems.

4. Algebraic Dynamic Logic

In this section, we present algebraic dynamic logic (*ADL*), in which algebraic transition systems are modeled as modalities by modifying first-order dynamic logic. Properties about the behavior of algebraic transition systems can be expressed as *ADL* formulas. After introducing the syntax of algebraic programs and formulas, we define an algebraic semantics of *ADL*, according to algebraic transition systems as mentioned in Section 3.

4.1. Syntax. The formulas of *ADL* are strings built over a finite set $\mathbf{V} = \{x_1, \ldots, x_n\}$ of real-valued variables and a signature Sig consisting of function symbols, predicate symbols, and constant symbols. In algebraic dynamic logic, modalities are extended to *algebraic programs* which are the combination of algebraic assertions and operational connectives.

Definition 12 (algebraic programs). The set of *algebraic programs* $\mathrm{Prm}(\mathbf{V}, \mathrm{Sig})$ is defined inductively as follows.

(i) If ψ is an algebraic assertion on $\mathbf{V} \cup \mathbf{V}'$ defined in Definition 8, then the *assignment* $\psi \in \mathrm{Prm}(\mathbf{V}, \mathrm{Sig})$ is an algebraic program.

(ii) If ψ is an algebraic assertion on $\mathbf{V}$, then the *guard* $\psi? \in \mathrm{Prm}(\mathbf{V}, \mathrm{Sig})$ is an algebraic program.

(iii) If α and β are algebraic programs, then the *sequential composition* $\alpha; \beta \in \mathrm{Prm}(\mathbf{V}, \mathrm{Sig})$.

(iv) If α and β are algebraic programs, then the *nondeterministic choice* $\alpha \cup \beta \in \mathrm{Prm}(\mathbf{V}, \mathrm{Sig})$.

(v) If α is an algebraic program, then the *iteration* $\alpha^* \in \mathrm{Prm}(\mathbf{V}, \mathrm{Sig})$.

As previously mentioned, the effect of an assignment $\psi \overset{\text{def}}{=} \bigwedge_i (f_i(x_1, \ldots, x_n, x_1', \ldots, x_n') = 0)$ is specified as a transition relation of algebraic transition systems. Furthermore, each f_i of ψ simultaneously takes place to change the current state. Assignments in computer programming languages are special cases of algebraic assertions since each next-state variable can easily be written as a unique polynomial in current-state variables according to assignment statements. The guard $\psi?$ is used to check whether the subsequent transition is possible. For the guard of $(\psi?; \alpha)$, the program α is allowed to happen, only when ψ is satisfied in the current state. Not all programs need a guard. Any program without a guard always takes place. The program $\alpha; \beta$ says that β is executed after doing α. The program $\alpha \cup \beta$ means that one of α and β is nondeterministically chosen and executed, and the program α^* says that α is executed some finite number of times.

Due to the operational structure of programs in standard dynamic logic [15], an algebraic transition system can be translated into an algebraic program without effort. Algebraic programs encode algebraic transition systems into modalities of *ADL* formulas, which specify properties of algebraic transition systems according to the following definition.

Definition 13 (formulas). The set of *ADL* formulas $\mathrm{Frm}(\mathbf{V}, \mathrm{Sig})$ is obtained inductively as follows.

(i) If $f(x_1, \ldots, x_n) \in \mathbb{R}[x_1, \ldots, x_n]$ is a polynomial defined over $\mathbf{V}$, then $f(x_1, \ldots, x_n) = 0 \in \mathrm{Frm}(\mathbf{V}, \mathrm{Sig})$ is an *atomic formula*.

(ii) If $\phi, \varphi \in \mathrm{Frm}(\mathbf{V}, \mathrm{Sig})$, then $\neg\phi, (\phi \wedge \varphi), (\phi \vee \varphi), (\phi \rightarrow \varphi) \in \mathrm{Frm}(\mathbf{V}, \mathrm{Sig})$.

(iii) If $\phi \in \mathrm{Frm}(\mathbf{V}, \mathrm{Sig})$ and $x \in \mathbf{V}$, then $\forall x\, \phi, \exists x\, \phi \in \mathrm{Frm}(\mathbf{V}, \mathrm{Sig})$.

(iv) If $\phi \in \mathrm{Frm}(\mathbf{V}, \mathrm{Sig})$ and $\alpha \in \mathrm{Prm}(\mathbf{V}, \mathrm{Sig})$, then $[\alpha]\phi, \langle\alpha\rangle\phi \in \mathrm{Frm}(\mathbf{V}, \mathrm{Sig})$.

The existential quantification can be defined with universal quantification and $\exists x\, \phi$ is abbreviated to $\neg\forall x\ \neg\phi$. The relation between $[\alpha]\phi$ and $\langle\alpha\rangle\phi$ is $\langle\alpha\rangle\phi \overset{\text{def}}{=} \neg[\alpha]\neg\phi$. The formula $[\alpha]\varphi$ expresses that all runs of program α lead to the states on which the formula φ holds. Likewise, $\langle\alpha\rangle\phi$ means that there exists at least a run of program α after which the formula φ holds. As for $[\alpha]\phi$ and $\langle\alpha\rangle\phi$, the algebraic program α plays the role of encoding an algebraic transition system, while ϕ claims that behavior of the algebraic transition system satisfies the property specified by ϕ. For example, the formula $\langle x' = 0; (x' = x + 1)^*\rangle\ (x - 3 = 0)$ asserts that there exists a run during the loop of $x' = x + 1$ such that x reaches 3 eventually.

Variables occurring in the scope of the quantifiers $\forall$ and $\exists$ are *bound to quantifiers*, and variables of $\mathbf{V}'$ occurring in modalities are *bound to modalities*. Variables are *free* if they are not in the scope of quantifiers and modalities. We assume that all variables are not bound to both of quantifiers and modalities at the same time. The interaction of quantifiers $\forall$, $\exists$ and modalities $[\cdot]$, $\langle\cdot\rangle$ makes the formulas subtle. Particularly, the order of quantifiers occurring before and after modalities makes the understanding of formulas slightly different. For instance, $\forall x[x' = x]\phi$ means that all the choices of the parameter x valued to x' keep ϕ true. However, for $[x' = x]\forall x\phi$, the variable x in $[x' = x]$ is free and different from x in $\forall x\phi$ which is a parameter and can be substituted with another variable symbol not occurring in ϕ. The way of unifying quantification and modalities in [15] is using a special *wildcard assignment* to redefine quantification such that $\forall x\phi \leftrightarrow [x' = ?]\phi$ and $\exists x\phi \leftrightarrow \langle x' = ?\rangle\phi$, where the wildcard assignment $x' = ?$ indicates an arbitrary assignment to x.

Example 14. We formalize the train control system shown in Figure 1 into the following algebraic program:

$$\text{train} \overset{\text{def}}{=} ((\text{Ac} \cup \text{Dc}); \text{drive})^*, \tag{15}$$

where

$$\begin{aligned} \text{Ac} &\overset{\text{def}}{=} (\text{mode} - \text{Acc} = 0)?; (v - 10 = 0)?; \\ &\quad a' + a = 0 \wedge \text{mode}' - \text{Dec} = 0 \\ \text{Dc} &\overset{\text{def}}{=} (\text{mode} - \text{Dec} = 0)?; (v - 5 = 0)?; \\ &\quad a' + a = 0 \wedge \text{mode}' - \text{Acc} = 0 \\ \text{drive} &\overset{\text{def}}{=} \left(v' - v - a\Delta t = 0\right) \wedge \left(s' - s - v\Delta t = 0\right). \end{aligned} \tag{16}$$

We use nondeterministic choice $\cup$ to join Ac and Dc together. In the phase of Ac, it tests whether the current mode is Acc and then checks whether the current velocity reaches 10. If so, the mode switches to Dec and the acceleration power a is inverted. The subsequent action is executing drive in which the velocity v and the position z evolve along $v' - v - a\Delta t = 0$ and $s' - s - v\Delta t = 0$, respectively. The phase of Dc is similar to Ac. The control system repeats (Ac $\cup$ Dc; drive) for indefinitely many times (or forever).

Furthermore, we express properties of the train control system as *ADL* formulas. For instance, the following statement about the train control system "the velocity of the train never reaches 11" is equivalently expressed as the formula

$$[\text{train}] \neg (v - 11 = 0). \tag{17}$$

4.2. Semantics. The semantics of *ADL* is defined in the fashion of Kripke [25], where possible worlds represent states of algebraic transition systems and transition relations along the runs of algebraic transition systems are represented as the accessibility relation.

For the set $\mathbf{V}$ and signature Sig, an *interpretation* $\mathcal{I}$ is a map, which maps each function symbol in Sig to an algebraic assertion on $\mathbf{V} \cup \mathbf{V}'$ and each predicate symbol in Sig to an algebraic assertion on $\mathbf{V}$. A *state* is a map $s : \mathbf{V} \rightarrow \mathbb{R}$ assigning a real value in $\mathbb{R}$ to each *state variable* in $\mathbf{V}$ whose value is only changed by algebraic programs. The free variables in $\mathbf{V}$ are mapped to the reals by an *assignment* $\zeta : \mathbf{V} \rightarrow \mathbb{R}$. These variables are also named *logical variables*. There is no need to distinguish logical variables and state variables except for the clarity of expressions.

The semantics of an algebraic program is interpreted as a transition relation consisting of pairs of states, while the satisfaction of an *ADL* formula is interpreted as a Boolean value by a state with respect to an interpretation and an assignment. We begin with the semantics of algebraic programs.

Definition 15 (semantics of algebraic programs). For each algebraic program $\alpha \in \text{Prm}(\mathbf{V}, \text{Sig})$, its semantics, denoted by $\gamma(\alpha)$, specifies the state w which is reachable from the state v under the operation of α. $\gamma(\alpha)$ is inductively defined as follows.

(i) $\gamma(\alpha) = \{(v, w) \mid (v, w) \in \text{Zero}(\psi)\}$ if $\alpha \overset{\text{def}}{=} \psi$, where ψ is an algebraic assertion on $\mathbf{V} \cup \mathbf{V}'$.

(ii) $\gamma(\alpha) = \{(v, v) \mid v \in \text{Zero}(\psi)\}$ if $\alpha \overset{\text{def}}{=} \psi?$, where ψ is an algebraic assertion on $\mathbf{V}$.

(iii) $\gamma(\alpha; \beta) = \{(v, w) \mid (v, z) \in \gamma(\alpha)$ and $(z, w) \in \gamma(\beta)$, for some state $z\}$.

(iv) $\gamma(\alpha \cup \beta) = \gamma(\alpha) \cup \gamma(\beta)$.

(v) $(v, w) \in \gamma(\alpha^*)$ if and only if there is a $n \in \mathbb{N}$ and a state sequence $v = v_0, \ldots, v_n = w$, with $(v_i, v_{i+1}) \in \gamma(\alpha)$ for all $0 \leq i < n$.

Note that the semantics of an algebraic program is defined according to zero sets of the algebraic assertion. Let $\alpha \overset{\text{def}}{=} \psi$ be an algebraic program with the algebraic assertion $\psi \overset{\text{def}}{=} \bigwedge_i f_i = 0$ on $\mathbf{V} \cup \mathbf{V}'$; the semantics of α is the common zero set of all f_i. For example, the semantics of $(x+1)^2 - (x')^2 = 0$ is the set of points lying on the two lines $x' - x - 1 = 0$ and $x' + x + 1 = 0$ in the $x - x'$ plane. There may exist more than one successive state for the current state. In most cases, the successive states of a given state are uniquely determined by algebraic programs. The guards in the form of $\psi?$ are associated with those states which satisfy the algebraic assertion ψ for triggering the next program. An iteration α^* points out all states reachable from the state v by successively executing α nondeterministically many times (zero or more).

The satisfaction of an *ADL* formula ϕ involves an interpretation $\mathcal{I}$, an assignment ζ, and a state v. For a formula ϕ, we write $\mathcal{I}, \zeta, v \models \phi$ and say that v and ζ satisfy ϕ in $\mathcal{I}$ or that ϕ is true in state v with respect to $\mathcal{I}$ and ζ. We omit $\mathcal{I}$, ζ and write $v \models \phi$ when $\mathcal{I}$ and ζ are understood in the context. The notation $v \not\models \phi$ means that v does not satisfy ϕ. We use $\zeta[x \mapsto c]$ to denote the *modification* of the assignment ζ that agrees with ζ except for the variable x which is amended to $c \in \mathbb{R}$.

Definition 16 (satisfaction of formulas). For two *ADL* formulas ϕ and φ, the satisfaction is inductively defined according to the syntactic structure of ϕ and φ.

(1) $\mathcal{I}, \zeta, v \models f(x_1, \ldots, x_n) = 0$ if and only if $v \in \text{Zero}(f)$.

(2) $\mathcal{I}, \zeta, v \models \neg\phi$ if and only if $\mathcal{I}, \zeta, v \not\models \phi$.

(3) $\mathcal{I}, \zeta, v \models \phi \wedge \varphi$ if and only if $\mathcal{I}, \zeta, v \models \phi$ and $\mathcal{I}, \zeta, v \models \varphi$. Similarly, for $\phi \vee \varphi$, $\phi \rightarrow \varphi$.

(4) $\mathcal{I}, \zeta, v \models \forall x\, \phi$ if and only if $\mathcal{I}, \zeta[x \mapsto c], v \models \phi$ for all $c \in \mathbb{R}$.

(5) $\mathcal{I}, \zeta, v \models \exists x\, \phi$ if and only if $\mathcal{I}, \zeta[x \mapsto c], v \models \phi$ for some $c \in \mathbb{R}$.

(6) $\mathcal{I}, \zeta, v \models [\alpha]\phi$ if and only if $\mathcal{I}, \zeta, w \models \phi$ for all states w with $(v, w) \in \gamma(\alpha)$.

(7) $\mathcal{I}, \zeta, v \models \langle\alpha\rangle\phi$ if and only if $\mathcal{I}, \zeta, w \models \phi$ for some state w with $(v, w) \in \gamma(\alpha)$.

A formula ϕ is *valid* in $\mathcal{I}$ and written as $\mathcal{I} \models \phi$ if ϕ is true on all states and all assignments in interpretation $\mathcal{I}$. If $\mathcal{I} \models \phi$ for all interpretations $\mathcal{I}$, we write $\models \phi$ and say that ϕ is *valid*.

After giving the semantics of algebraic dynamic logic, in order to prove the validity of *ADL* formulas, such as (17), we construct a proof system for *ADL* in the next section.

5. Proof System

In this section, we construct a sequent calculus for algebraic dynamic logic. In a sequent calculus a *sequent* is an expression of the form $\Gamma \vdash \Delta$, where the *antecedent* $\Gamma = (\phi_1, \ldots, \phi_m)$ and the *succedent* $\Delta = (\varphi_1, \ldots, \varphi_n)$ are finite sequences of formulas. The meaning of $\Gamma \vdash \Delta$ is equivalently expressed as the following formula:

$$\phi_1 \wedge \cdots \wedge \phi_m \longrightarrow \varphi_1 \vee \cdots \vee \varphi_n. \tag{18}$$

That is to say, a sequent $\Gamma \vdash \Delta$ is satisfied by a state v if and only if $v \models \phi_1 \wedge \cdots \wedge \phi_m \rightarrow \varphi_1 \vee \cdots \vee \varphi_n$. Equivalently, v makes the sequent false if v makes $\phi_1, \ldots, \phi_m$ all true and $\varphi_1, \ldots, \varphi_n$ all false.

An *inference rule* is of the form

$$\frac{S_1 \cdots S_i}{S'}, \tag{19}$$

where both $S_1 \cdots S_i$ and S' are sequents. The upper sequents $S_1 \cdots S_i$ are called *premises* and the lower sequent S' is called *conclusion*. The semantics of an inference rule is that each state satisfying all premises also makes the conclusion true. The direction of entailment is top-down which means that premises logically imply the conclusion, while the direction of applying rules is bottom-up. That means that the procedure of reasoning about a sequent starts from the conclusion at the bottom to the premises at the top.

The proof system, called *ADL* calculus, is constructed by customizing inference rules which manipulate *ADL* formulas in an algebraic fashion. The basic idea is evaluating the effects of algebraic programs with algebraic methods mentioned in Section 2 and transforming *ADL* formulas into first-order formulas without algebraic programs. *ADL* calculus consists of axiom rules, rules for logical operators, rules for quantifier, rules for modalities, and programs.

5.1. Rules for Axioms. In this and the following sections, the symbols Γ, Δ denote arbitrary sequences of *ADL* formulas and ϕ, φ denote *ADL* formulas unless otherwise noted.

Four basic rules (A_1)–(A_4) listed in (T1), named *axiom rules*, are composed for closing a proof search. Rules (A_1)–(A_3) are the same as in many other sequent calculus. The axiom rule (A_1) treats a sequent with a common formula in the antecedent and the succedent as an axiom, which can be inferred from nothing (denoted by $\perp$):

$$(A_1)\ \frac{\perp}{\Gamma, \phi \vdash \Delta, \phi} \quad (A_2)\ \frac{\perp}{\Gamma \vdash \text{True}, \Delta} \qquad (A_3)\ \frac{\perp}{\Gamma, \text{False} \vdash \Delta} \quad (A_4)\ \frac{\perp}{\Gamma, \phi \vdash \varphi, \Delta}. \tag{T1}$$

In A_4, ϕ and φ are atomic formulas which are expressed as polynomials in variables of $\mathbf{V}$, and $\text{Zero}(\phi) \subseteq \text{Zero}(\varphi)$.

Rule (A_4) is customized to coordinate mathematical procedures on ideal theory implemented by computer algebra systems, such as REDUCE, Maple, Mathematica, AXIOM, and SINGULAR. Rule (A_4) reveals that any sequent whose antecedent has a formula ϕ with its zero set included by the zero set of one formula φ of the succedent can be applied as an axiom. By Theorems 5 and 6, the inclusion $\text{Zero}(\phi) \subseteq \text{Zero}(\varphi)$ can be transformed into the radical membership problem which is decided by *radical membership algorithm* on polynomial ideals [5, 24]. That is to say, if there is an atomic formula φ in the succedent such that φ belongs to the radical of the ideal generated by polynomials in ϕ in the antecedent, that is, $\varphi \in \sqrt{\langle\phi\rangle}$, then the sequent can be applied as an axiom. The radical membership algorithm is implemented in most computer algebra systems, such as the `RadicalMembership` command in Maple. The discussion on computer algebra systems is not in the scope of this paper.

5.2. Rules for Logical Operators. The rules in (T2) are used to handle standard logical operators. There are two cases for the appearance of each logical operator, and each logical operator needs dual rules (left rule and right rule):

$$\begin{aligned}
&(L_1)\ \frac{\Gamma \vdash \phi, \Delta}{\Gamma, \neg\phi \vdash \Delta} && (L_2)\ \frac{\Gamma, \phi \vdash \Delta}{\Gamma \vdash \neg\phi, \Delta} \\
&(L_3)\ \frac{\Gamma, \phi, \varphi \vdash \Delta}{\Gamma, \phi \wedge \varphi \vdash \Delta} && (L_4)\ \frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \varphi, \Delta}{\Gamma \vdash \phi \wedge \varphi, \Delta} \\
&(L_5)\ \frac{\Gamma, \phi \vdash \Delta \quad \Gamma, \varphi \vdash \Delta}{\Gamma, \phi \vee \varphi \vdash \Delta} && (L_6)\ \frac{\Gamma \vdash \phi, \varphi, \Delta}{\Gamma \vdash \phi \vee \varphi, \Delta} \\
&(L_7)\ \frac{\Gamma, \varphi \vdash \Delta \quad \Gamma \vdash \phi, \Delta}{\Gamma, \phi \longrightarrow \varphi \vdash \Delta} && (L_8)\ \frac{\Gamma, \phi \vdash \varphi, \Delta}{\Gamma \vdash \phi \longrightarrow \varphi, \Delta}.
\end{aligned} \tag{T2}$$

Rules (L_1)–(L_8) are standard for propositional dynamic logic. These rules decompose formulas with propositional structures into smaller formulas with less logical operators. Rules (L_1) and (L_2) are dual and aim to reduce the negative operator $\neg$. Rule (L_3) just replaces the symbol $\wedge$ with a comma, since formulas are combined conjunctively in antecedents of sequents by the definition of sequents. Rule (L_4) branches the sequent containing the operator $\wedge$ in the succedent into two sequents, since conjuncts in the succedent can be proved separately due to the semantics of sequents. Dually, rule (L_5) is similar to (L_4) and (L_6) is similar to (L_3) according to the semantics of sequents. Rule (L_7) derives from rules (L_5) and (L_1) by the logical equivalence of $(\phi \rightarrow \varphi)$ and $(\neg\phi \vee \varphi)$. Similarly, rule (L_8) is derived from rules (L_6) and (L_2).

5.3. Rules for Quantifiers. Recall that variables of $\mathbf{V}'$ occurring in an algebraic program are *bound* to modalities. We assume that each modality-bound variable is not bound to any quantifier. A variable is free if it is not bound to modalities and quantifiers.

Definition 17 (substitution). A *substitution* of an algebraic assertion or a formula for a free variable is defined as a function which maps each object variable to a designated polynomial. Let f be a polynomial; the result of substituting f in an algebraic assertion ψ for a variable x is denoted by $\psi[x \mapsto f]$. The result of substituting f in a formula ϕ for a variable x is denoted by $\phi[x \mapsto f]$.

A substitution with the result $\phi[x \mapsto f]$ is *admissible* for the formula ϕ if no variables y in f are bound in the formula $\phi[x \mapsto f]$. That is to say, free variables in f are still free in the formula $\phi[x \mapsto f]$ after applying an admissible substitution.

Rule (Q_1) and rule (Q_4) are the usual γ-rule in [7, 26], while (Q_2) and (Q_3) correspond to the δ^+-rule in [26]. Rules (Q_1)–(Q_4) are complete for quantifiers of classic first-order logic. That is to say, everything about quantifiers of first-order logic can be derived with (Q_1)–(Q_4) [7]:

$$\begin{aligned}
&(Q_1)\ \frac{\Gamma, \phi[x \longmapsto X], \forall x\phi \vdash \Delta}{\Gamma, \forall x\phi \vdash \Delta} && (Q_2)\ \frac{\Gamma \vdash \phi[x \longmapsto f(X_1, \ldots, X_n)], \Delta}{\Gamma \vdash \forall x\phi, \Delta} \\
&(Q_3)\ \frac{\Gamma, \phi[x \longmapsto f(X_1, \ldots, X_n)] \vdash \Delta}{\Gamma, \exists x\phi \vdash \Delta} && (Q_4)\ \frac{\Gamma \vdash \phi[x \longmapsto X], \exists x\phi, \Delta}{\Gamma \vdash \exists x\phi, \Delta}.
\end{aligned} \tag{T3}$$

In (Q_1)–(Q_4), the variable x is only bound to $\forall x$ or $\exists x$. In (Q_1) and (Q_4), X is a new free variable. In (Q_2) and (Q_3), f is a new function symbol and $X_1, \ldots, X_n$ are all free variables in $\forall x \phi$ and $\exists x \phi$.

5.4. Rules for Modalities and Programs. The rules for modalities and programs are obtained from the rule schemata shown in (T4), which can be applied in both sides of a sequent. Rule schemata in (T4) analyze the effect of modalities by reducing algebraic programs into simpler ones.

If

$$\frac{\phi_1}{\phi_0} \tag{20}$$

is an instance of one rule schema in (T4), then

$$\frac{\Gamma, \phi_1 \vdash \Delta}{\Gamma, \phi_0 \vdash \Delta}, \qquad \frac{\Gamma \vdash \phi_1, \Delta}{\Gamma \vdash \phi_0, \Delta} \tag{21}$$

are two inference rules of the *ADL* calculus. There is one rule schema for each program structure $(;, ?, \cup, *)$ and modalities $(\langle\cdot\rangle, [\cdot])$.

As mentioned previously, algebraic programs are defined on $\mathbf{V} \cup \mathbf{V}'$. *ADL* formulas, which are defined on $\mathbf{V}$, only assert properties on the final states of the runs of algebraic programs. Confusions about variable may emerge when an algebraic program needs to be lifted to a formula by rules in (T4). For eliminating the confusion, variables of algebraic programs need be renumbered by the *variable numbering procedure* defined by the following definition when an algebraic program is lifted to a formula with rule (M_1) and rule (M_2).

Definition 18 (variable numbering procedure). For each sequent $\Gamma \vdash \Delta$, there always exists a procedure such that a sequent, which does not produce any confusion about variable, is obtained from $\Gamma \vdash \Delta$ by numbering all occurrences of variables.

Since every sequent is assumed to contain finite variables, numbering finite variables is easy and immediately leads to a variable numbering procedure. We assume that each sequent, which produces variable confusions, is implicitly numbered by the variable numbering procedure. For the sake of succinctness, the description of this procedure is not shown in detail. However, the effect of this procedure is illustrated by Example 19:

$$\begin{array}{llll}
(M_1) \ \dfrac{\forall X_\Psi (\Psi \longrightarrow \phi)}{[\Psi]\, \phi} & (M_2) \ \dfrac{\exists X_\Psi (\Psi \wedge \phi)}{\langle\Psi\rangle\, \phi} \\[2ex]
(M_3) \ \dfrac{\Psi \longrightarrow \phi}{[\Psi?]\, \phi} & (M_4) \ \dfrac{\Psi \wedge \phi}{\langle\Psi?\rangle\, \phi} \\[2ex]
(P_1) \ \dfrac{[\alpha]\, [\beta]\, \phi}{[\alpha;\beta]\, \phi} & (P_2) \ \dfrac{\langle\alpha\rangle\, \langle\beta\rangle\, \phi}{\langle\alpha;\beta\rangle\, \phi} \\[2ex]
(P_3) \ \dfrac{[\alpha]\, \phi \wedge [\beta]\, \phi}{[\alpha \cup \beta]\, \phi} & (P_4) \ \dfrac{\langle\alpha\rangle\, \phi \vee \langle\beta\rangle\, \phi}{\langle\alpha \cup \beta\rangle\, \phi} \\[2ex]
(P_5) \ \dfrac{\phi \wedge [\alpha;\alpha^*]\, \phi}{[\alpha^*]\, \phi} & (P_6) \ \dfrac{\phi \vee \langle\alpha;\alpha^*\rangle\, \phi}{\langle\alpha^*\rangle\, \phi}.
\end{array} \tag{T4}$$

In (M_1) and (M_2), Ψ denotes an algebraic assertion on $\mathbf{V} \cup \mathbf{V}'$. In (M_3) and (M_4), Ψ denotes an algebraic assertion on $\mathbf{V}$. Variables bound to Ψ are denoted by X_Ψ.

Example 19. Consider the sequent

$$\vdash \left[x^2 - (x')^2 - 4 = 0; x^2 - (x')^2 - 2 = 0\right] (x - 2 = 0). \tag{22}$$

It would have produced confusion of the variables x and x' if we applied rules (M_1) and (M_2) directly. After doing the variable numbering procedure, we get the following sequent:

$$\vdash \left[x_0^2 - x_1^2 - 4 = 0; x_1^2 - x_2^2 - 2 = 0\right] \quad (x_2 - 2 = 0), \tag{23}$$

which produces no confusions when applying M_1 and M_2.

Rules (M_1) and (M_2) are used to deal with assignment programs in $[\cdot]$ and $\langle\cdot\rangle$. The basic idea of (M_1) and (M_2) is transforming an *ADL* formula with assignment programs into a standard first-order formula by lifting assignment programs to logical formulas. (M_1) expresses that the formula ϕ always holds after executing the assignment program Ψ, if Ψ implies ϕ for all values of variables bound to Ψ while (M_2) expresses that ϕ holds for some execution of Ψ, if both Ψ and ϕ hold for some value for variables bound to Ψ. Both (M_1) and (M_2) properly reflect the underlying logical principle that *ADL* formulas with assignment programs can be transformed into quantified formulas. The rules (M_3) and (M_4) can be understood in the same way as in [15], except that the logical formula of test is replaced with a guard specified by an algebraic assertion Ψ on $\mathbf{V}$.

Rules (P_1)–(P_6) are used to decompose the structure of programs into simpler programs. In order to prove the sequential compositions of programs, nested modalities, which are obtained by decomposing sequential compositions, have to be proved by (P_1)-(P_2). Nondeterministic choices are proved by proving the conjunction by (P_3) or disjunction by (P_4) of its alternatives. (P_5) and (P_6) are the usual iteration rules in dynamic logic [15], which unfold loops.

5.5. Miscellaneous Rules. Besides the rules mentioned previously, some miscellaneous rules are necessary to our proof system. There are several types of rules listed in (T5). The first type is the usual generalization rules (G_1 and G_2) which allow to derive $[\alpha]\phi \vdash [\alpha]\varphi$ and $\langle\alpha\rangle\phi \vdash \langle\alpha\rangle\varphi$ from $\vdash \phi \rightarrow \varphi$.

(C_1) is the usual cut rule [27] which does not make our proof system prove more theorems but just allows the proofs to be shorter and simpler. (C_1) states that when a formula ϕ can be concluded in the context and ϕ can also serve as a premise for concluding other formulas, then the formula ϕ can be cut out from the context. However, when searching a proof bottom-up with the cut rule, it requires one to guess the auxiliary formula ϕ.

Rule (I_1) is a variant of the usual induction rule with the *inductive invariant* ϕ [15]. It expresses that the invariant ϕ will be true after any number of iterations of α, if ϕ is true in the current state and ϕ is still true after the execution of α when

$$
I_1\,\dfrac{L_4\,\dfrac{L_2\,\dfrac{v-11=0\vdash}{\vdash\neg(v-11=0)}\qquad \dfrac{\Pi}{\vdash\forall X_\alpha(\neg(v-11=0)\longrightarrow[\mathrm{Ac}\cup\mathrm{Dc};\mathrm{drive}]\neg(v-11=0))}}{\vdash\neg(v-11=0)\wedge\forall X_\alpha(\neg(v-11=0)\longrightarrow[\mathrm{Ac}\cup\mathrm{Dc};\mathrm{drive}]\neg(v-11=0))}}{\vdash[(\mathrm{Ac}\cup\mathrm{Dc};\mathrm{drive})^*]\neg(v-11=0)}
$$

FIGURE 2: Verification of the train control system.

$$
Q_2\,\dfrac{L_8\,\dfrac{P_1\,\dfrac{P_3\,\dfrac{\dfrac{\Pi_2}{\neg(v-11=0)\vdash[\mathrm{Ac}][\mathrm{drive}]\neg(v-11=0)}\qquad\Pi_1}{\neg(v-11=0)\vdash[\mathrm{Ac}\cup\mathrm{Dc}][\mathrm{drive}]\neg(v-11=0)}}{\neg(v-11=0)\vdash[\mathrm{Ac}\cup\mathrm{Dc};\mathrm{drive}]\neg(v-11=0)}}{\vdash\neg(v-11=0)\longrightarrow[\mathrm{Ac}\cup\mathrm{Dc};\mathrm{drive}]\neg(v-11=0)}}{\vdash\forall X_\alpha(\neg(v-11=0)\longrightarrow[\mathrm{Ac}\cup\mathrm{Dc};\mathrm{drive}]\neg(v-11=0))}
$$

FIGURE 3: Proof of the right branch.

ϕ holds for all bound variables of α. Rule (O_1) transforms a finite set of atomic formulas with an empty zero set into the Boolean value False. Whether atomic formulas have an empty zero set is decided by Theorem 7. Consider

$$
\begin{array}{llll}
(G_1) & \dfrac{\vdash\phi\longrightarrow\varphi}{[\alpha]\phi\vdash[\alpha]\varphi} & (G_2) & \dfrac{\vdash\phi\longrightarrow\varphi}{\langle\alpha\rangle\phi\vdash\langle\alpha\rangle\varphi} \\
(C_1) & \dfrac{\Gamma\vdash\phi,\Delta\quad\Gamma,\phi\vdash\Delta}{\Gamma\vdash\Delta} & (I_1) & \dfrac{\phi\wedge\forall X_\alpha(\phi\longrightarrow[\alpha]\phi)}{[\alpha^*]\phi} \\
(O_1) & \dfrac{\text{False}\vdash}{\Gamma,\phi_1,\ldots,\phi_n\vdash} & &
\end{array}
\quad (\mathrm{T5})
$$

X_α in (I_1) denotes all variables bound to the algebraic program α. In rule (O_1) formulas $\phi_1,\ldots,\phi_n$ are atomic such that they have an empty zero set.

6. Verification Example

Reasoning about the safety property of the train control system, which is formulated by (17), is shown in Figures 2–4. Remark that we have assumed the parameter Δt is constant; that is, the time is discrete and modelled by Δt. In addition, the train evolves its velocity per Δt and keeps the velocity fixed in each Δt. We start by writing the safety property $[\mathrm{train}]\neg(v-11=0)$ into the proof obligation:

$$\vdash[\mathrm{train}]\neg(v-11=0). \qquad (24)$$

We use the rule (I_1) to eliminate the iteration operator and split the sequent into two branches by applying rule (L_4). The left branch derives the open sequent $v-11=0\vdash$ from $\vdash\neg(v-11=0)$ by applying (L_2). Recall that by the semantics of sequent the proof obligation is valid if all premises are valid. Hence (24) is not valid if $(v-11=0)\vdash$ is not valid. In other words,

$$v\neq 11 \qquad (25)$$

which makes $(v-11=0)\vdash$ valid must be initially guaranteed in order to make (24) valid.

$$
M_3\,\dfrac{L_8\,\dfrac{P_1,M_3\,\dfrac{L_8\,\dfrac{P_1,M_1\,\dfrac{Q_2\,\dfrac{L_8,L_3\,\dfrac{M_1\,\dfrac{Q_2\,\dfrac{L_7,L_3\,\dfrac{L_2,L_1\,\dfrac{\Gamma_0,\Gamma_1,\Gamma_2,v'=11\vdash v-11=0}{\Gamma_0,\Gamma_1,\Gamma_2,\neg(v-11=0)\vdash\neg(v'=11)}}{\Gamma_0,\Gamma_1,\neg(v-11=0)\vdash\phi_3}}{\Gamma_0,\Gamma_1,\neg(v-11=0)\vdash\forall v'\forall s'\phi_3}}{\Gamma_0,\Gamma_1,\neg(v-11=0)\vdash[\mathrm{drive}]\neg(v-11=0)}}{\Gamma_0,\neg(v-11=0)\vdash\phi_2}}{\Gamma_0,\neg(v-11=0)\vdash\forall a'\forall\mathrm{mode}'\phi_2}}{\Gamma_0,\neg(v-11=0)\vdash\phi_1}}{\neg(v-11=0),\mathrm{mode}=\mathrm{Acc}\vdash v=10\longrightarrow\phi_1}}{\neg(v-11=0),\mathrm{mode}=\mathrm{Acc}\vdash\phi_0}}{\neg(v-11=0)\vdash\mathrm{mode}=\mathrm{Acc}\longrightarrow\phi_0}}{\neg(v-11=0)\vdash[\mathrm{Ac}][\mathrm{drive}]\neg(v-11=0)}
$$

FIGURE 4: The rest of proof.

The proof of the right branch (Π) goes on as shown in Figure 2. Since there are no free variables in $\forall X_\alpha(\neg(v-11=0)\rightarrow[\mathrm{Ac}\cup\mathrm{Dc};\ \mathrm{drive}]\neg(v-11=0))$, we replace each bound variable v with a constant v when the rule (Q_2) is applied. Proving the sequent $\neg(v-11=0)\vdash[\mathrm{Ac}\cup\mathrm{Dc}][\mathrm{drive}]\neg(v-11=0)$ is to prove both $\neg(v-11=0)\vdash[\mathrm{Ac}][\mathrm{drive}]\neg(v-11=0)$ and $\neg(v-11=0)\vdash[\mathrm{Dc}][\mathrm{drive}]\neg(v-11=0)$ (denoted by Π_1 in Figure 2).

The proof of the sequent $\neg(v-11=0)\vdash[\mathrm{Ac}][\mathrm{drive}]\neg(v-11=0)$ (Π_2) is shown in Figure 3. For notional convenience, the notions Γ_0, Γ_1, are Γ_2 are introduced as follows:

$$
\begin{aligned}
&\Gamma_0\overset{\mathrm{def}}{=}v=10,\quad\mathrm{mode}=\mathrm{Acc}\\
&\Gamma_1\overset{\mathrm{def}}{=}a'=-a,\quad\mathrm{mode}'=\mathrm{Dec}\\
&\Gamma_2\overset{\mathrm{def}}{=}s'=s+v\Delta t,\quad v'=v+a'\Delta t\\
&\phi_0\overset{\mathrm{def}}{=}\left[v=10?;a'=-a\wedge\mathrm{mode}'=\mathrm{Dec}\right][\mathrm{drive}]\\
&\quad\cdot\neg(v-11=0)\\
&\phi_1\overset{\mathrm{def}}{=}\left[a'=-a\wedge\mathrm{mode}'=\mathrm{Dec}\right][\mathrm{drive}]\\
&\quad\cdot\neg(v-11=0)\\
&\phi_2\overset{\mathrm{def}}{=}a'=-a\wedge\mathrm{mode}'=\mathrm{Dec}\\
&\quad\longrightarrow[\mathrm{drive}]\neg(v-11=0)\\
&\phi_3\overset{\mathrm{def}}{=}\left(v'=v+a'\Delta t\right)\wedge\left(s'=s+v\Delta t\right)\\
&\quad\longrightarrow\neg\left(v'=11\right).
\end{aligned}
\qquad (26)
$$

At the end of the proof in Figure 3, we obtain the sequent

$$\Gamma_0, \Gamma_1, \Gamma_2, v' = 11 \vdash v - 11 = 0, \tag{27}$$

where v denotes the initial velocity and v' denotes the final velocity after executing program [Ac ∪ Dc; drive]. Since we have known that the initial velocity v does not equal 11 according to (25), the right side of (27) is false. Hence the left side of (27) should be unsatisfiable to make sequent (27) valid. If we treat a, v, and Δt as parameters of the train, then $v - a\Delta t - 11 = 0$ is concluded from $\Gamma_0 \wedge \Gamma_1 \wedge \Gamma_2 \wedge v' = 11$ by eliminating variables except $a, v, \Delta t$. If the assertion $v - a\Delta t - 11 = 0$ is not satisfied, sequent (27) will be valid. That means

$$v - a\Delta t \neq 11 \tag{28}$$

implies $\neg(v - 11 = 0) \vdash [\mathrm{Ac}][\mathrm{drive}]\neg(v - 11 = 0)$.

The proof of the sequent Π_1 is similar to Π_2. In the same way, we get the same assertion:

$$v - a\Delta t \neq 11 \tag{29}$$

which implies $\neg(v - 11 = 0) \vdash [\mathrm{Dc}][\mathrm{drive}]\neg(v - 11 = 0)$. Hence, the safety property on the train control system specified by (17) is valid if the requirements specified by (25), (28), and (29) are satisfied; that is,

$$v \neq 11 \wedge (v - a\Delta t \neq 11) \Longrightarrow [\mathrm{train}] \neg (v - 11 = 0). \tag{30}$$

7. Soundness and Incompleteness

7.1. Soundness. In this section, we prove that the *ADL* calculus is sound; that is, verification with the *ADL* calculus always produces correct results.

Lemma 20. *Given a formula ϕ, two interpretations $\mathcal{I}$, $\mathcal{J}$, assignments ζ, η, and states v, w, if $\mathcal{I}, \zeta, v$ and $\mathcal{J}, \eta, w$ agree on all free variables of ϕ, then $\mathcal{I}, \zeta, v \vDash \phi$ if and only if $\mathcal{J}, \eta, w \vDash \phi$.*

Proof. This lemma can be proved by a simple induction principle [28] on the structures of algebraic programs and *ADL* formulas as in Section 4.2 using the semantics of algebraic programs and *ADL* formula satisfaction. □

The soundness of the *ADL* calculus is shown by the following theorem, which says that all rules preserve the validity of sequents.

Theorem 21. *For all rules and rule schemata listed in* (T1)–(T5), *if all premises of a rule are valid sequents which are true in all states for all interpretations, then the conclusion is a valid sequent.*

Proof. We start by giving the proof of special rules for *ADL* calculus such as (A_4), (M_1), (M_2), (O_1), and (I_1). The rest of the rules are proved in an immediate way which can be easily found in [15, 26, 28] and thus are skipped over here.

(i) Rule (A_4) is a sound rule for axioms. Let $\mathcal{I}$ be an interpretation, ζ an assignment, and v a state with $\mathcal{I}, \zeta, v \vDash \phi \wedge \Gamma$. Since the zero set of ϕ in the antecedent of the conclusion is a subset of the zero set of φ in the succedent, that is, $\mathrm{Zero}(\phi) \subseteq \mathrm{Zero}(\varphi)$, we can conclude that $v \in \mathrm{Zero}(\phi) \subseteq \mathrm{Zero}(\varphi)$. Then we conclude that $v \vDash \varphi \vee \Delta$ by the definition of $\vDash$. According to the semantics of sequents $\Gamma, \phi \vdash \varphi, \Delta$ is true for all $\mathcal{I}, \zeta, v$ and can be applied as an axiom.

(ii) Rule (Q_4) is sound. For any $\mathcal{I}$, ζ, v with $\mathcal{I}, \zeta, v \vDash \phi[x \mapsto X]$, we choose $\zeta(X)$ as the witness of $\exists x\, \phi$ such that the value of X in ζ is assigned to the bound variable x in $\exists x\, \phi$. Then $\mathcal{I}, \zeta, v \vDash \exists x\, \phi$ can be concluded. The proof of (Q_1) is trivial since the premise and the conclusion contain the common formula $\forall x\, \phi$.

(iii) The soundness proof of (Q_3) is shown as follows. Assume that there are $\mathcal{I}$, v such that for some ζ the formula $\phi[x \mapsto f(X_1, \ldots, X_n)]$ is true; that is, $\mathcal{I}, \zeta, v \vDash \phi[x \mapsto f(X_1, \ldots, X_n)]$. We can conclude that $\mathcal{I}, \zeta, v \vDash \exists x\, \phi$ by choosing the value $\zeta(f(X_1, \ldots, X_n))$ of $f(X_1, \ldots, X_n)$ as the witness of $\exists x\, \phi$. The soundness proof of (Q_2) is dual.

(iv) (M_1) is sound. Assume that there are $\mathcal{I}$, v such that for all assignments ζ the formula Ψ implies ϕ; that is, $\mathcal{I}, \zeta, v \vDash \forall X_\Psi\ (\Psi \rightarrow \phi)$. Since all bound variables in the modality $[\Psi]$ are denoted by X_Ψ, we can conclude that $\mathcal{I}, \zeta, w \vDash \phi$ for all w with $(v, w) \in \gamma(\Psi)$ by the definition of semantics of $[\cdot]$ and Lemma 20. Then $\mathcal{I}, \zeta, v \vDash [\Psi]\phi$ is concluded.

(v) (M_2) is sound. Assume that there are $\mathcal{I}$, v such that for some assignment ζ the formula $\Psi \wedge \phi$ is true; that is, $\mathcal{I}, \zeta, v \vDash \exists X_\Psi\ (\Psi \wedge \phi)$. We can conclude that $(v, w) \in \gamma(\Psi)$ with $w = v[X_\Psi \mapsto \zeta(X_\Psi)]$ by the definition of $\gamma(\Psi)$. Then $\mathcal{I}, \zeta, w \vDash \phi$ is concluded by Lemma 20. Hence we can conclude that $\mathcal{I}, \zeta, v \vDash \langle \Psi \rangle \phi$.

(vi) (O_1) is sound. Since the zero set of atomic formulas $\phi_1, \ldots, \phi_n$ is empty, there does not exist a state v such that $v \vDash \Gamma \wedge \phi_1 \wedge \cdots \wedge \phi_n$ by the satisfaction of formulas. So we can conclude that the antecedent is unsatisfiable and is equivalent to False.

(vii) Rule schema (I_1) is sound. For any $\mathcal{I}$, ζ, v with $\mathcal{I}, \zeta, v \vDash \phi \wedge \forall X_\alpha (\phi \rightarrow [\alpha]\phi)$, we conclude that $\mathcal{I}, \zeta, w \vDash \phi$ for all states w with $(v, w) \in \gamma(\alpha^*)$ by Lemma 20. Hence we can further conclude that $\mathcal{I}, \zeta, v \vDash [\alpha^*]\phi$ by the semantics of $[\alpha^*]$.

□

7.2. Incompleteness. The following theorem shows the inherent incompleteness of *ADL* calculus.

Theorem 22. *The ADL calculus is inherently incomplete.*

Proof. This theorem is proved by showing that the Peano arithmetic producing natural numbers is definable in *ADL*. Natural numbers are definable in modalities using iterations:

$$\mathrm{Nat}(n) \equiv \left\langle x' = 0; \left(x' = x + 1\right)^* \right\rangle x = n. \tag{31}$$

Hence the incompleteness theorem of Gödel [29] can be applied in *ADL* calculus. Hence we immediately conclude that the *ADL* calculus is incomplete. □

8. Conclusions and Future Work

In this paper, we present a deductive approach for reasoning about algebraic transition system. This approach models algebraic transition systems as algebraic programs of *ADL*, which is obtained by allowing algebraic assertions in dynamic logic. The properties of algebraic transition systems are formalized as *ADL* formulas with our method. We explain the semantics of algebraic programs in *ADL* as transition relations of algebraic transition systems and define the satisfaction of *ADL* formulas zero sets of polynomials. A proof system for *ADL*, called *ADL* calculus, is constructed for reasoning about algebraic transition systems. The *ADL* calculus is proved to be sound and is illustrated by the verification of the safety property of the train control system.

Our approach combines mathematical procedures on polynomial ideal theory with the deductive verification by customizing special rules for handling algebraic programs. The introduction of mathematical procedures enhances the reasoning power of our proof system. However proofs of properties related to iterations and quantifiers may be tedious and ineffective in complex cases. Future work includes a closer investigation for effective rules of iterations and quantifiers, for example, the invariant method [4, 30–32] and quantifiers elimination [33–35]. On the other hand, there are properties which cannot be formalized as the *ADL* formulas such as properties with inequalities, since *ADL* formulas are defined with algebraic assertions which actually are polynomial equations. In addition to the future work, more general *ADL* formulas should include inequalities and more complex structures, such as differential equations for specifying hybrid systems.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive comments and suggestion. This work is supported by the National Natural Science Foundation of China under Grants nos. 11371003 and 11461006, the Natural Science Foundation of Guangxi under Grants nos. 2011GXNSFA018154, 2012GXNSFGA060003, and 2014GXNSFAA118359, the Science and Technology Foundation of Guangxi under Grant no. 10169-1, and the Scientific Research Project no. 201012MS274 from Guangxi Education Department.

References

[1] J.-P. Katoen, "Lablelled transition systems," in *Model-Based Testing of Reactive Systems*, vol. 3472 of *Lecture Notes in Computer Science*, chapter 29, pp. 615–616, Springer, Berlin, Germany, 2005.

[2] V. D'Silva, D. Kroening, and G. Weissenbacher, "A survey of automated techniques for formal software verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1165–1178, 2008.

[3] R. M. Keller, "Formal verification of parallel programs," *Communications of the ACM*, vol. 19, no. 7, pp. 371–384, 1976.

[4] S. Sankaranarayanan, H. B. Sipma, and Z. Manna, "Non-linear loop invariant generation using grobner bases," *ACM SIGPLAN Notices*, vol. 39, no. 1, pp. 318–329, 2004.

[5] D. A. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, vol. 10, Springer, 2007.

[6] L. Lamport, "The temporal logic of actions," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 3, pp. 872–923, 1994.

[7] M. Fitting, "First-order logic and automated theorem proving," *Studia Logica*, vol. 61, no. 2, pp. 300–302, 1998.

[8] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, 1999.

[9] C. Baier and J.-P. Katoen, *Principles of Model Checking*, MIT Press, Cambridge, Mass, USA, 2008.

[10] T. A. Henzinger, R. Majumdar, and J.-F. Raskin, "A classification of symbolic transition systems," *ACM Transactions on Computational Logic*, vol. 6, no. 1, pp. 1–32, 2005.

[11] A. Chutinan and B. H. Krogh, "Verification of infinite-state dynamic systems using approximate quotient transition systems," *IEEE Transactions on Automatic Control*, vol. 46, no. 9, pp. 1401–1410, 2001.

[12] A. Platzer, "Differential dynamic logic for hybrid systems," *Journal of Automated Reasoning*, vol. 41, no. 2, pp. 143–189, 2008.

[13] B. Beckert and S. Schlager, "A sequent calculus for first-order dynamic logic with trace modalities," in *Automated Reasoning*, vol. 2083 of *Lecture Notes in Computer Science*, pp. 626–641, Springer, Berlin, Germany, 2001.

[14] E. M. Clarke and J. M. Wing, "Formal methods: state of the art and future directions," *ACM Computing Surveys*, vol. 28, no. 4, pp. 626–643, 1996.

[15] D. Harel, D. Kozen, and J. Tiuryn, *Dynamic Logic*, Foundations of Computing Series, MIT Press, Cambridge, Mass, USA, 2000.

[16] F. Kroger and S. Merz, "Verification of state systems," in *Temporal Logic and State Systems*, Texts in Theoretical Computer Science, pp. 213–256, Springer, Berlin, Germany, 2008.

[17] S. Merz, "The specification language TLA$^+$," in *Logics of Specification Languages*, Monographs in Theoretical Computer Science, chapter 8, pp. 401–451, Springer, Berlin, Germany, 2008.

[18] K. Chaudhuri, D. Doligez, L. Lamport, and S. Merz, "Verifying safety properties with the TLA$^+$ proof system," in *Automated Reasoning*, J. Giesl and R. Hahnle, Eds., vol. 6173 of *Lecture Notes in Computer Science*, pp. 142–148, Springer, Berlin, Germany, 2010.

[19] D. Cousineau, D. Doligez, L. Lamport et al., "TLA$^+$ proofs," in *Proceedings of the 18th International Symposium on Formal Methods (FM '12)*, G. Dimitra and M. Dominique, Eds., vol. 7436, pp. 147–154, Springer, 2012.

[20] Z. Manna and A. Pnueli, *Temporal Verification of Reactive Systems: Safety*, vol. 2, Springer, New York, NY, USA, 1995.

[21] Z. Manna and H. B. Sipma, "Deductive verification of hybrid systems using STeP," in *Hybrid Systems: Computation and Control*, vol. 1386 of *Lecture Notes in Computer Science*, pp. 305–318, Springer, Berlin, Germany, 1998.

[22] A. R. Bradley and Z. Manna, *The Calculus of Computation: Decision Procedures with Applications to Verification*, Springer, Berlin, Germany, 2007.

[23] A. Platzer, "The complete proof theory of hybrid systems," in *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LICS '12)*, pp. 541–550, IEEE Computer Society, Los Alamitos, Calif, USA, 2012.

[24] B. Mishra, *Algorithmic Algebra*, Springer, 1993.

[25] S. A. Kripke, "Semantical considerations on modal logic," *Acta Philosophica Fennica*, vol. 16, pp. 83–94, 1963.

[26] R. Hähnle and P. H. Schmitt, "The liberalized δ-rule in free variable semantic tableaux," *Journal of Automated Reasoning*, vol. 13, no. 2, pp. 211–221, 1994.

[27] C. Dunchev, A. Leitsch, M. Rukhaia, and D. Weller, "Cut-elimination and proof schemata," in *Logic, Language, and Computation*, M. Aher, D. Hole, E. Jeřábek, and C. Kupke, Eds., vol. 8984 of *Lecture Notes in Computer Science*, pp. 117–136, Springer, Berlin, Germany, 2015.

[28] J. H. Gallier, *Logic for Computer Science: Foundations of Automatic Theorem Proving*, Harper & Row Publishers, 2003.

[29] R. M. Smullyan, *Gödel's Incompleteness Theorems*, Oxford University Press, 1992.

[30] S. Sankaranarayanan, H. B. Sipma, and Z. Manna, "Constructing invariants for hybrid systems," in *Hybrid Systems: Computation and Control*, vol. 2993 of *Lecture Notes in Computer Science*, pp. 539–554, Springer, Berlin, Germany, 2004.

[31] R. Rebiha, A. Moura, and N. Matringe, "Generating invariants for non-linear loops by linear algebraic methods," *Formal Aspects of Computing*, 2015.

[32] A. R. Bradley and Z. Manna, "Property-directed incremental invariant generation," *Formal Aspects of Computing*, vol. 20, no. 4-5, pp. 379–405, 2008.

[33] L. González-Vega, "A combinatorial algorithm solving some quantifier elimination problems," in *Quantifier Elimination and Cylindrical Algebraic Decomposition*, B. Caviness and J. Johnson, Eds., Texts and Monographs in Symbolic Computation, pp. 365–375, Springer, Vienna, Austria, 1998.

[34] G. Pan and M. Y. Vardi, "Symbolic techniques in satisfiability solving," *Journal of Automated Reasoning*, vol. 35, no. 1–3, pp. 25–50, 2005.

[35] V. Weispfenning, "A new approach to quantifier elimination for real algebra," in *Quantifier Elimination and Cylindrical Algebraic Decomposition*, B. Caviness and J. Johnson, Eds., Texts and Monographs in Symbolic Computation, pp. 376–392, Springer, Vienna, Austria, 1998.