Approximated Model Checking for Multirate Hybrid ZIA

Guozheng Li, Zining Cao and Zheng Gao

Computer Science Department Nanjing University of Aeronautics & Astronautics,
Nanjing, China
leebug38@foxmail.com

Abstract

Virtually all control systems nowadays perform various behavioral aspects such as discrete control mode transformation and continuous real time behavior. The interaction of these different types of dynamics and information leads to a lot of safety and control problems. In this paper, to verify these systems we propose a specification model combining interface automata, initialized multirate hybrid automata and Z language, named MZIA. This model can be used to describe temporal properties, hybrid properties, and data properties of hybrid software/hardware complements. And then, we propose a temporal logic for MZIA. Next, considering the measuring errors of real-numbered variables in practice, we study the approximated model checking method of MZIA. Finally, an example is given to indicate that this method is feasible and effective.

Keywords: Control Systems, Interface Automata, Z Notation, Hybrid Automata, Approximated Model Checking

1. Introduction

Most of complicated control systems like flight control, manufacturing systems and transportation exhibit various behavioral aspects such as discrete and continuous transition, communication between components, and state transformation inside components. To ensure the correctness of these “hybrid control” systems, formal specification techniques for such systems have to be able to describe all these aspects. Unfortunately, a single specification technique that is well suited for all these aspects is yet not available. Instead one needs various specialized techniques that are very good at describing individual aspects of system behavior. This observation has led to research into the combination and semantic integration of specification techniques. In this paper we combine three well researched specification techniques: Interface automata, multirate hybrid automata and Z.

Interface automaton is a light-weight automata-based language for component specification, which was proposed in [1]. An interface automaton (IA), introduced by de Alfaro and Henzinger, is an automata-based model suitable for specifying component-based systems. Hybrid automaton [2] is a formal model for a mixed discrete-continuous system. Z [3] is a typed formal specification notation based on first order predicate logic and set theory. Model checking [4, 5] is a method of verifying concurrent systems in which a state-graph model of the system behavior is compared with a temporal logic formula. But it is only for verifying finite state concurrent systems. One benefit of this restriction is that verification can be performed automatically.

In [6, 7], we have proposed the ZIA and HZIA model, but didn’t give the corresponding temporal logic and the model checking algorithm. In this paper, we present a new specification language which combines interface automata, multirate hybrid automata and Z language. Interface automata are a kind of intuitive models for interface property of software components. Multirate hybrid automata are a model of mixed discrete-continuous systems. Z can describe the data property of states and transitions of a
system. To specify mixed discrete-continuous software/hardware components, we give the definition of MZIA. Roughly speaking, a MZIA is in a style of hybrid interface automata but its states and operations are described by Z language. Furthermore, a logic called Real Timed-Data Constraints Logic for MZIAs is defined. And, considering the measuring errors of real-numbered variables in practice, an algorithm for approximated model checking this logic for MZIAs with finite domain is provided.

This paper is organized as follows: In Section 2, we propose a specification language-MZIA. In Section 3, logic for MZIA is proposed. And we give the definition of MZIA with finite domain in section 4. In section 5, we give an approximated model checking algorithm for MZIA with finite domain. In section 6, we indicate the procedure of the model checking algorithm by an example. The paper is concluded in Section 7.

2. Multirate Hybrid Interface Automata with Z

In many cases, systems have both discrete and continuous property. To specify hybrid systems, we proposed the specification ZIA and HZIA in [6, 7]. In [8], the author proved the model checking of initialized multirate hybrid automata to be decidable. So for the decidability of our model checking algorithm, in this paper we add some constrains to HZIA, named MZIA, which can be used to specify hybrid behavioral and the data structure aspects of a system as well.

Let \( X = \{x_1, \ldots, x_n\} \) be a set of real-numbered variables, \( \mathbb{R} \) all real numbers, and \( \mathbb{Q} \) all rational numbers. A rectangle \( B \subseteq \mathbb{R}^n \) over \( X \) is defined by a conjunction of linear (in) equalities of the form \( x_i - c \), where \( c \in \mathbb{Q} \), \( x_i \in X \), and \( \varepsilon \in \{<,\leq,=,\geq\} \). We use \( B_i \) to denote its projection onto the \( i \)th coordinate and \( \mathcal{R}(X) \) the set of all \( n \)-dimensional rectangles. More details of multirate hybrid automata can be referred to the article [9].

**Definition 1.** A multirate hybrid interface automata with Z (MZIA) \( P = (S_p, S_p', A_p', A_p^*, A_p^0, A_p^U, X_p, V_p^I, V_p^O, V_p^U, C_p, F_p^S, F_p^A, I_P, T_P) \) consists of the following elements:

1. \( S_p \) is a set of states;
2. \( S_p' \subseteq S_p \) is a set of initial states. If \( S_p' = \emptyset \) then \( P \) is called empty;
3. \( A_p', A_p^0 \) and \( A_p^U \) are disjoint sets of input, output, and internal actions, respectively. We denote by \( A_p = A_p' \cup A_p^0 \cup A_p^U \) the set of all actions;
4. \( X_p = \{x_1, \ldots, x_n\} \) is a finite set of real-numbered variables; The number \( n \) is called the dimension of \( P \). We write \( \dot{X}_p = \{x'_1, \ldots, x'_n\} \) representing first derivatives during continuous change, and \( \ddot{X}_p = \{x''_1, \ldots, x''_n\} \) representing values at the conclusion of discrete change. \( \mathcal{R}(X_p) \) is a rectangle over \( X_p \);
5. \( V_p^I, V_p^O \) and \( V_p^U \) are disjoint sets of input, output, and internal variables, respectively. We denote by \( V_p = V_p^I \cup V_p^O \cup V_p^U \) the set of all variables. We have that \( X_p \subseteq V_p \) which are all continuous valued variables and \( V_p - X_p \) are all discrete valued variables;
6. \( C_p \) is a variable representing time, whose value is a real number, \( C_p \notin V_p \);
7. \( F_p^S \) is a map, which maps any state in \( S_p \) to a state schema \( \Phi(V_p \cup \{C_p\}) \) in Z language;
8. \( F_p^A \) is a map, which maps any input action in \( A_p' \) to an input operation schema \( \Phi(V_p) \) in Z language, and maps any output action in \( A_p^0 \) to an output operation...
schema $\Phi(V_p)$ in Z language, and maps any internal action in $A_p^u$ to an internal operation schema $\Phi(V_p)$ in Z language;

(9) $I_p$ is a tuple $(inv_p, init_p, act_p)$, mapping from any state in $S_p$ to $\square^n$ or $\square(X_p)$, where $init_p : S_p \rightarrow \square^n$ assigns an initial condition to each state, $inv_p : S_p \rightarrow \square(X_p)$ assigns an invariant condition to each state, and $act_p : S_p \rightarrow \square^n$ assigns a flow condition to each state $s \in S_p$ to indicate that $\dot{x} = act_p(s)$, for each $x \in X_p$;

(10) $T_p \subseteq S_p \times A_p \times \square(X_p) \times \square^2 \times \square^n \times S_p$ is a set of transitions. The 6-tuple $(s, a, \varphi, \lambda, \xi, s') \in T_p$ corresponds to a transition from state $s$ to state $s'$ labeled with action $a \in A_p(s)$, a constraint $\varphi$ that specifies when the transition is enabled, and a set of real-numbered variables $\lambda \subseteq \square^2$ that are reset to the corresponding value in $\xi$ when the transition is executed. In this paper, we define that if for every coordinate $i \in \{1, \ldots, m\}$ with $act_p(s) \neq act_p(s')$, then $x_i \in \lambda$. Furthermore, we have that $| = (F^e(s') \cap F^e(a)) \setminus (x_1, \ldots, x_m)$

(11) $\Leftrightarrow F^e_p(t)[y'_1, y'_1, \ldots, y'_n, y_n]$, where $\{x_1, \ldots, x_n\}$ is the set of variables in $F^e_p(s)$, $\{y_1, \ldots, y_n\}$ is the set of the variables in $F^e_p(t)$, the set variables in $F^e_p(a)$ is the subset of $\{x_1, \ldots, x_m\} \cup \{y_1, \ldots, y_n\}$.

3. Logic for MZIA

In this section, we present a temporal logic for MZIAs. Traditional methods (CTL, LTL) for reasoning about reactive system abstract away from quantitative time preserving only qualitative properties (such as “eventually p holds”) [10]. In [10], the author proposed a logic called TCTL. Here, we extend TCTL to RT-DCL (Real Timed-Data Constraints Logic) to reason about both timing behavioral and the data structure in a MZIA.

3.1. Syntax

Throughout this paper, we let RT-DCL be a language which is just the set of formulas of interest to us.

Definition 2. The set of formulas called RT-DCL is given by the following rules:

(1) If $\phi$ is in the form of $p(x_1, \ldots, x_n)$, then $\phi \in RT-DCL$, where $p$ is a $n$-ary prediction, $x_1, \ldots, x_n$ are variables. In this paper, we define that a state predicate $p$ should be expressed by a rectangle over the set $\{x_1, \ldots, x_n\} \subseteq \{x_1, \ldots, x_n\}$, where $\{x_1, \ldots, x_n\}$ is the real-numbered variables set;

(2) If $\phi_1, \phi_2 \in RT-DCL$, then $\phi_1 \land \phi_2 \in RT-DCL$;

(3) If $\phi_1, \phi_2 \in RT-DCL$, then $\phi_1 \lor \phi_2 \in RT-DCL$;

(4) If $\phi \in RT-DCL$, then $\forall x : T \phi \in RT-DCL$;

(5) If $\phi \in RT-DCL$, then $\exists x : T \phi \in RT-DCL$;

(6) If $\phi \in RT-DCL$, then $\text{EX} \phi \in RT-DCL$;

(7) If $\phi \in RT-DCL$, then $\text{AX} \phi \in RT-DCL$;

(8) If $\phi \in RT-DCL$, then $c \cdot \phi \in RT-DCL$. $c \cdot \phi$ is a reset operator, where $c$ is clock variable;

(9) If $\phi_1, \phi_2 \in RT-DCL$, then $E \phi_1 \cup \phi_2 \in RT-DCL$, where $\sim \{<, \leq, =, \neq, >, \geq\}$ and $c \in \square$;

(10) If $\phi_1, \phi_2 \in RT-DCL$, then $A \phi_1 \cup \phi_2 \in RT-DCL$, where $\sim \{<, \leq, =, \neq, >, \geq\}$ and $c \in \square$.

(11) If $\phi \in RT-DCL$, then $\text{EG}_c \phi \in RT-DCL$, where $\sim \{<, \leq, =, \neq, >, \geq\}$ and $c \in \square$;
(12) If $\phi \in RT-DCL$, then $AG_{c}\phi \in RT-DCL$, where \( \sim \in \{<,\leq,=,\neq,>,\geq\} \) and $c \in \mathbb{N}$.

### 3.2. Semantics

We will describe the semantics of $RT-DCL$, that is, whether a given formula is true or false. Since MZIA have some real-numbered variables which may be measured with small errors. We should consider the measuring errors of real-numbered variables in the description of the semantics of $RT-DCL$.

In the following, we use $AV(A)$ to denote the set of all variables in Z schema $A$, and $CV(A)$ to denote the set of real-numbered variables in Z schema $A$. In order to define that Z schemas satisfy $RT-DCL$ formulas approximately, we need the following notation.

**Definition 3.** Given a positive real-numbered assignment $\delta$ on $\{x_1,\ldots,x_m\}$ which represents the measuring errors of real-numbered variables $\{x_1,\ldots,x_m\}$, an assignment $\rho$ on $\{y_1,\ldots,y_n\}$, where $\{x_1,\ldots,x_m\}$ are set of all real-numbered variables in Z schema $\{y_1,\ldots,y_n\}$. We use the notation $\rho \oplus \delta$ to denote the set of all assignments $\{\sigma(y) = \rho(y) \mid y \notin \{x_1,\ldots,x_m\}\}$, and $\sigma(x) = \rho(x) + \delta(x)$, where $\delta(x) \leq \alpha \leq \delta(x)$. We use notation $\rho \oplus \delta$ to denote $\{\sigma \mid \sigma \in CV(\sigma(y))\}$.

**Definition 4.** Given MZIA $P$, a computation in $P$ is a possibly infinite sequence of states $\pi = (s_0, s_1, \cdots)$ if there is an $a_0$ such that $(s_0, a_0, s_{i+1}) \in T_c$ for each $i \in \mathbb{N}$, i.e., $s+1$ is the state of the performing an $a$ on the state $s$. For a computation $\pi = (s_0, s_1, \cdots)$, let $\pi[k] = s_k$, and $\pi_i = (s_0, s_1, \cdots, s_i)$ for each $k \in \mathbb{N}$, where $\mathbb{N}$ is all natural numbers. By $\prod(s)$ we denote the set of all the infinite computations starting at $s$ in $P$.

**Definition 5.** Semantics of $RT-DCL$:

Given a MZIA $P$, a $RT-DCL$ formula $\phi$ and $s \in S_P$, we define the satisfaction relation $(P,s) \models \phi$ inductively:

1. $(P,s) \models \phi$ if and only if $\phi$ is true in state $s$.
2. $(P,s) \models \phi \land \psi$ if $(P,s) \models \phi$ and $(P,s) \models \psi$.
3. $(P,s) \models \phi \lor \psi$ if $(P,s) \models \phi$ or $(P,s) \models \psi$.
4. $(P,s) \models \exists x:T \phi$ if $(P,s) \models \phi$.
5. $(P,s) \models \exists x:T \phi$ if $(P,s) \models \phi$.
6. $(P,s) \models \text{EX} \phi$ if there exists a state $s' \in S_P$ such that $(s,a,s') \in T_c$ and $(P,s') \models \phi$.
7. $(P,s) \models \text{AX} \phi$ if for each state $s' \in S_P$ and $(s,a,s') \in T_c$, we have $(P,s') \models \phi$.
8. $(P,s) \models c \phi$ if $(P,s)_{c=0} \models \phi$, where $(P,s)_{c=0}$ means that the clock variable is set to zero, and $(P,s)_{c=0} \models \phi$ means that $\phi$ is still true in state $(P,s)$ after $c$ is reset to zero.
9. $(P,s) \models \text{EX} \phi \ U \phi$ if there exists a computation $\pi \in \prod(s)$ such that $\pi[t] \models \phi$, where $t \sim c$, and $\pi[t'] \models \phi$ for each $t' \in (0,t)$.
(10) \((P, s) \models A \phi U _\sim \phi\), iff for each computation \(\pi \in \prod (s)\), we have \(\pi [t] \models \phi\), where \(t \sim c\), and \(\pi [t'] \models \phi\) for each \(t' \in (0, t)\).

(11) \((P, s) \models EG _\sim \phi\), iff there exists a computation \(\pi \in \prod (s)\) such that \(\pi [t] \models \phi\), where \(t \sim c\);

(12) \((P, s) \models AG _\sim \phi\), iff for each computation \(\pi \in \prod (s)\), we have \(\pi [t] \models \phi\), where \(t \sim c\).

4. MZIA with Finite Domain

Consider a MZIA \(P\) and a pair \((s, D_p) \in S_p \times \mathbb{R}\), where \(\mathbb{R}\) is the set of real numbers. Obviously, MZIA \(P\) is an infinite state system. While model checking is a technique for verifying finite state systems, we should first convert infinite-state to finite-state. To obtain a finite representation for infinite state space of MZIA, we give the definition of MZIA with finite domain in this section. Roughly speaking, in a MZIA, each state and each action are assigned to a schema in Z language. If every discrete variable in any schema has finite possible values and continuous variables are represented by multirate zones, such MZIAs are called MZIAs with finite domain.

4.1. Multirate Zones

In [11], the author proposed a constraint system called multirate zone for the representation and manipulation of multirate hybrid automata state-spaces. A multirate zone is a conjunction of inequalities of the following types: \(ax \leq by + c\), \(x < c\), and \(c < x\), where \(\in \{<,\leq\}\), \(c \in \mathbb{R}\). Furthermore, the author showed that a multirate zone can be represented by a difference constraint matrix (DCM) and also gave three operations on DCMs: intersection, variable reset, and elapsing of time and proved that DCMs keep closed to the three operations.

We use multirate zone as the basis for the infinite state-space exploring of multirate hybrid automata, as well as for MZIAs. To realize the multirate zones in the computer expediently, we use the DCM structure. In section 6, we describe the exploring process with an example.

4.2. MZIA with Finite Domain

Here we will introduce a class of MZIAs, for which model checking problem is decidable.

**Definition 6.** Given a Z schema \(\bar{s} [v_1: T_1; \ldots; v_m: T_m | P_1; \ldots; P_p]\), we call it a Z schema with finite domain, if every discrete variable \(v_i\) in any schema has finite possible value, i.e., each type \(T_i\) has finite elements.

**Definition 7.** A MZIA \(P = \{S_p, S_p^0, A_p^0, A_p^u, X_p, V_p^0, V_p^u, C_p, F_p, \Gamma_p, I_p, T_p\}\) is called a MZIA with finite domain, if the following condition holds:

1. For each \(s \in S_p\), \(F_p^s (s)\) is a Z schema with finite domain;
2. For each \(a \in S_p\), \(F_p^a (a)\) is a Z schema with finite domain.

As multirate hybrid automata can be represented by DCM, so we get the finite state-space of MZIA with finite domain easily.

5. Model Checking Algorithm for MZIA with Finite Domain

In this section, we propose the approximated model checking algorithm for MZIAs. In [12], Rajeev Alur proposed a labeling algorithm and gave the correctness-proof of the
algorithm. Here we improve his algorithm with adding data constraints. And also, we give our thoughts on the implementation of our model checking procedure.

5.1. Algorithm

The basic idea for our algorithm is as follows: Given a MZIA $P$ and a $RT-DCL$ formula $f$, at first we convert $P$ to a MZIA with finite domain, denoted by $F(P)$. Then, we label the zones in $F(P)$ with subformulas of $f$, starting from the subformulas of length 1, then of length 2, and so on. If the initial states of $F(P)$ are labeled with $f$, then $F(P)$ satisfies formula $f$ approximately.

In the previous section, we represent multirate zones by difference constraint matrix. So it is easy for us to realize the process of MZIA $P$ converting to MZIA with finite domain $F(P)$. Next, approximated model checking for MZIA with finite domain $F(P)$ is our major consideration.

Our algorithm will operate by labeling each state $s$ with the set $label(s)$ of subformulas of $f$ which are true in $s$. Initially, $label(s)$ is just atomic proposition formula set which are true in $s$. The algorithm then goes through a series of stages. During the $i$th stage, subformulas with $i-1$ nested $RT-DCL$ operators are processed. Once a subformula is processed, we add it to the labeling of each state in which it is true. At last, the algorithm terminates, and we will have that $(F(P), s) \models f$ iff $f \in label(s)$.

We define the function $Sub$, when given a formula $\phi$, returns a queue of syntactic subformulas of $\phi$ such that if $\phi_1$ is a subformula of $\phi$ and $\phi_2$ is a subformula of $\phi_1$, then $\phi_2$ precedes $\phi_1$ in the queue $Sub(\phi)$. Procedure for labeling the states of $F(P)$ satisfying some $RT-DCL$ formula $f$ is as follows:
For each $\Psi$ in Sub $(f)$ do:

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>case $\Psi = p(x_1,\ldots,x_n)$</td>
<td>if $F^\nu_p(s) \models p(x_1,\ldots,x_n)$ then $\text{label}(s) = \text{label}(s) \cup \Psi$; break; // where $Z$ schema $F^\nu_p(s)$ is regarded as a first order logical formula</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>case $\Psi = c[\phi]$</td>
<td>if $s \in { s \mid c = 0$ and $\phi \in \text{label}(s) }$ then $\text{label}(s) = \text{label}(s) \cup \Psi$; break; // where $c$ is a clock variable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>case $\Psi = \phi \lor \psi$</td>
<td>if $\phi \in \text{label}(s)$ or $\psi \in \text{label}(s)$ then $\text{label}(s) = \text{label}(s) \cup \Psi$; break;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>case $\Psi = (\forall x:T) \phi$</td>
<td>if $s \in \bigcap_{x \in T} { t \mid \phi \in \text{label}(t) }$ then $\text{label}(s) = \text{label}(s) \cup \Psi$; break;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>case $\Psi = (\exists x:T) \phi$</td>
<td>if $s \in \bigcap_{x \in T} { t \mid \phi \in \text{label}(t) }$ then $\text{label}(s) = \text{label}(s) \cup \Psi$; break;</td>
</tr>
</tbody>
</table>

Partial correctness of the algorithm can be proved induction on the structure of the input formula $f$. Termination is guaranteed, because the state space of $F(P)$ is finite. Therefore we have the following proposition:

**Proposition 1.** The algorithm given in the above terminates and is correct, i.e., it returns the set of subformulas of the input formula $f$ which are true in each state.

**Proof.** We can prove this proposition by induction on the structure of $\Psi$. Assume that $s$ is labeled with $\Psi$ iff $(F(P), s) \models \Psi$.

1. If $\Psi$ is atomic proposition formula $p(x_1,\ldots,x_n)$, and $s$ is labeled with $\Psi$, then $F^\nu_p(s) \models p(x_1,\ldots,x_n)$. So we have $(F(P), s) \models \Psi$;

2. If $\Psi = \phi \land \phi$, and $s$ is labeled with $\Psi$, then $s$ is also labeled with $\phi$ and $\phi$. By the induction hypothesis, $(F(P), s) \models \phi$ and $(F(P), s) \models \phi$. So we have $(F(P), s) \models \phi \land \phi$, i.e., $(F(P), s) \models \Psi$;

---

Copyright © 2016 SERSC
If $\Psi = \phi \lor \phi_i$ and $s$ is labeled with $\Psi$, then $s$ is labeled with either $\phi_i$ or $\phi$. By the induction hypothesis, $(F(P), s) \models \phi_i$ or $(F(P), s) \models \phi$. So we have $(F(P), s) \models \phi_0 \land \phi_1$, i.e., $(F(P), s) \models \Psi$;

If $\Psi = (\forall x : T) \phi$, and $s$ is labeled with $\Psi$, then for each $u \in T$, $s$ is labeled with $\phi[u/x]$. By the induction hypothesis, we have that $(F(P), s) \models \phi[u/x]$, for each $u \in T$, i.e., $(F(P), s) \models \Psi$;

If $\Psi = (\exists x : T) \phi$, and $s$ is labeled with $\Psi$, then for each $u \in T$, $s$ is labeled with $\phi[u/x]$. By the induction hypothesis, we have that $(F(P), s) \models \phi[u/x]$, for each $u \in T$, i.e., $(F(P), s) \models \Psi$;

If $\Psi = \text{EX} \phi$, and $s$ is labeled with $\Psi$, then there exists some state $s \in S_\rho$ and $(s, a, s') \in T_\rho$ such that $s'$ is labeled with $\phi$. By the induction hypothesis, we have that $(F(P), s) \models \text{EX} \phi$, i.e., $(F(P), s) \models \Psi$;

If $\Psi = \text{AX} \phi$, and $s$ is labeled with $\Psi$, then for each state $s' \in S_\rho$ and $(s, a, s') \in T_\rho$, we have $s'$ is labeled with $\phi$. By the induction hypothesis, we have that $(F(P), s) \models \text{AX} \phi$, i.e., $(F(P), s) \models \Psi$;

If $\Psi = \text{c} \phi$, and $s$ is labeled with $\Psi$, then $s$ is still labeled with $\phi_i$ after the clock variable $c$ is reset to 0. By the induction hypothesis, $(F(P), s)_{n \to 0} \models \phi$. So we have $(F(P), s) \models \Psi$;

If $\Psi$ is $E \phi U \phi_i$ and $s$ is labeled with $\Psi$, then there exists a computation $\pi = (s_0, s_1, \ldots, s_n) \in \prod(s)$ such that $s_i$ is labeled with $\phi_i$ for each $1 \leq i < n$, $s_n$ is labeled with $\phi$, and clock variable $z$ satisfies $z \sim c$. By the induction hypothesis, there exists a computation $\pi' = (s_0, s_1, \ldots, s_n) \in \prod(s)$ such that $(F(P), s_0) \models \phi_i$, $(F(P), s_n) \models \phi_0$ and $z \sim c$. So we have $(F(P), s) \models E(\phi U \phi_0)$, i.e., $(F(P), s) \models \Psi$;

If $\Psi$ is $A \phi U \phi_i$ and $s$ is labeled with $\Psi$, then every computation $\pi = (s_0, s_1, \ldots, s_n) \in \prod(s)$ satisfies that $s_i$ is labeled with $\phi_i$ for each $1 \leq i < n$, $s_n$ is labeled with $\phi$, and clock variable $z$ satisfies $z \sim c$. By the induction hypothesis, every computation $\pi = (s_0, s_1, \ldots, s_n) \in \prod(s)$ such that $(F(P), s_0) \models \phi_i$, $(F(P), s_n) \models \phi_0$ and $z \sim c$. So we have $(F(P), s) \models A(\phi U \phi_0)$, i.e., $(F(P), s) \models \Psi$.

If $\Psi$ is $E G_{\phi_0} \phi$, and $s$ is labeled with $\Psi$, then there exists a computation $\pi = (s_0, s_1, \ldots, s_n) \in \prod(s)$ such that $s_i$ is labeled with $\phi$ for each $1 \leq i \leq n$, and clock variable $z$ satisfies $z \sim c$. By the induction hypothesis, there exists a computation $\pi = (s_0, s_1, \ldots, s_n) \in \prod(s)$ such that $(F(P), s_i) \models \phi$ and $z \sim c$. So we have $(F(P), s) \models E G_{\phi_0} \phi$, i.e., $(F(P), s) \models \Psi$.

If $\Psi$ is $A G_{\phi_0} \phi$, and $s$ is labeled with $\Psi$, then every computation $\pi = (s_0, s_1, \ldots, s_n) \in \prod(s)$ satisfies that $s_i$ is labeled with $\phi$ for each $1 \leq i \leq n$, and clock variable $z$ satisfies $z \sim c$. By the induction hypothesis, every computation $\pi = (s_0, s_1, \ldots, s_n) \in \prod(s)$ satisfies $(F(P), s_i) \models \phi$, and $z \sim c$. So we have $(F(P), s) \models A G_{\phi_0} \phi$, i.e., $(F(P), s) \models \Psi$. 


5.2. Implementation

Most existing tools [13, 14] focus on model checking for the discrete-time or linear hybrid state space, which is not suitable for our model checking algorithm for MZIA. The main reasons are that:

- The MZIA model can describe data properties of hybrid system based on Z language. But existing tools seldom achieve this with Z language.
- The RT-DCL logic we proposed can describe both timing behavioral and the data structure in a MZIA. Also we consider the measuring errors of real-numbered variables in the logic. But logics in existing tools cannot make it.

In the following, we will concentrate on converting our research to practical application. The main ideas and data structures of our simple implementation are as follows.

1. We first give some enumerated types representing the action type, variable type, and connector. Then we construct the structure of action and variable.

   ```
   enum action_type { input_action, output_action, inter_action }; // action type
   struct Action{
       enum action_type a_type; // action type
       string name; // action name
   };
   struct Variable{
       string name; // variable name
       int value; // variable value
       enum var_type v_type; // variable type
       int low_range; // the range of variable
       int up_range;
   };
   ```

2. Next, binary tree is used to represent the logic formula. We introduce the variable error to describe the approximately satisfying of logic formulas.

   ```
   struct formula_node{
       enum symbol sym; // connector
       formula_node *parent; // point to parent node
       formula_node *left; // point to left child node
       formula_node *right; // point to right child node
       Variable var;
       double error; // approximate error
       int layer; // convenient for judging the logic formula
   };
   struct formula{
       formula_node *schema; // formula in the Z schema
       display();
   };
   ```

3. The structure of states denoted by Z schema is as follows.

   ```
   struct status{
       vector<Variable> val; // all variables
       formula var_formula; // formula of the state
   };
   ```

4. Here we represent the transition by edge, which includes action description. The start point and end point are also stored.

   ```
   struct edge_node { // transition between two states
       int star_point; // the start point
       int end_point; // the end point
       Action act;
       formula act_formula; // formula in action of Z schema
   };
   ```
(5) We need the following structure representing that the set of edges and nodes from one given state.

```c
struct sys_node{
  state s; // state
  vector<edge_node> e; // edge
};
```

(6) At last, we give the structure of our MZIA model.

```c
struct MZIA{
  vector<sys_node> node; // the set of states and transitions
  vector<int> InitState; // the set of initial states
};
```

Given a MZIA $P$, we first convert to MZIA with finite domain $F(P)$ by the $\text{succ}(M, e)$ operation. All the states and formulas in $F(P)$ are denoted by the data structures above. As described in our approximated model checking algorithm, we check whether $F(P)$ satisfies some formulas by recursively calling the sub-process. So we have a preliminary implementation of our research.

6. Example

In this section, we demonstrate the procedure for the approximated model checking with a simple example.

6.1. Boiler Plant Description

Boiler has been widely used in thermal power station, ships, industrial and mining enterprises and so on. In [15], the author model a steam-boiler control system using hybrid automata. Here we consider a simple boiler plant including temperature controller, boiler system and pressure monitor, as in Figure 1. For convenience, we only consider the temperature and pressure in boiler system, and consider temperature controller and pressure monitor as two components communicating with boiler system by some interfaces. To be specific, the temperature is controlled by the temperature controller and the pressure is automatically controlled by the valve. The boiler system will send the pressure value to the pressure monitor.

Given a MZIA $P$, we first convert to MZIA with finite domain $F(P)$ by the $\text{succ}(M, e)$ operation. All the states and formulas in $F(P)$ are denoted by the data structures above. As described in our approximated model checking algorithm, we check whether $F(P)$ satisfies some formulas by recursively calling the sub-process. So we have a preliminary implementation of our research.

6. Example

In this section, we demonstrate the procedure for the approximated model checking with a simple example.

6.1. Boiler Plant Description

Boiler has been widely used in thermal power station, ships, industrial and mining enterprises and so on. In [15], the author model a steam-boiler control system using hybrid automata. Here we consider a simple boiler plant including temperature controller, boiler system and pressure monitor, as in Figure 1. For convenience, we only consider the temperature and pressure in boiler system, and consider temperature controller and pressure monitor as two components communicating with boiler system by some interfaces. To be specific, the temperature is controlled by the temperature controller and the pressure is automatically controlled by the valve. The boiler system will send the pressure value to the pressure monitor.

Figure 2 illustrates some states transitions of temperature and pressure, which is a MZIA model. The variable $x$ and $y$ represent the temperature and pressure, respectively. $x?$ means the boiler system receive a input signal from the temperature controller, and $y!$ means the boiler system send a output signal to the pressure monitor. Initially, we suppose that the temperature is 20(°C) and the pressure is standard atmospheric pressure, i.e. 100(Kpa). We omit the unit in the following. The automaton has four locations. The
temperature and pressure are governed by derivatives in different location. The automaton starts in location \( l_0 \). It can remain in that location as long as the pressure is less than or equal to 1000. As soon as the pressure is greater than or equal to 700, the automaton can make a transition to location \( l_1 \) and reset the pressure to 700. Simultaneously, the derivative of the pressure is reset to 30. The rest of the transitions are similar.

### 6.2. DCM Representation

We now see how the construction of the zones transitions. Multirate zones are represented by difference constraint matrix and the successor state is computed by the three operations on difference constraint matrix described in [11].

\[
\begin{align*}
\text{Figure 2. States Transitions of Boiler System}
\end{align*}
\]

Firstly, the initial state is given by \( s_0 = (l_0, x? = 20 \land y! = 100) \) which corresponds to the difference constraint matrix \( M_0 \):

\[
\begin{align*}
x? & \quad y! \\
(1, 1, 0, \leq) & (20, 1, -20, \leq) & (20, 1, -100, \leq) \\
x? & (1, 20, 20, \leq) & (1, 1, 0, \leq) & (20, 20, -80, \leq) \\
y! & (1, 20, 100, \leq) & (20, 20, 80, \leq) & (1, 1, 0, \leq)
\end{align*}
\]

We show the operation steps of getting next state with the intersection, variable reset, and elapsing of time operations. Only the canonical form DCM [11] obtained in each step is shown.

(1) The invariant in location \( l_0 \) is \( \text{inv}(l_0) = y! \leq 1000 \), which is given by the matrix:

\[
\begin{align*}
x? & \quad y! \\
x_0 & \quad x? & \quad y! \\
(1, 1, 0, \leq) & (20, 1, \infty, \leq) & (20, 1, \infty, \leq) \\
x? & (1, 20, \infty, \leq) & (1, 1, 0, \leq) & (20, 20, \infty, \leq) \\
y! & (1, 20, 1000, \leq) & (20, 20, \infty, \leq) & (1, 1, 0, \leq)
\end{align*}
\]

(2) Next, we let time elapse in the location \( l_0 \) using the operator \( \tilde{} \). The matrix for \( (M_0 \land \text{inv}(l_0))\tilde{} \) is:
(3) The jump condition $\varphi = 700 \leq y$ for the $a_0$ transition from location $l_0$ to location $l_1$ is:

$$
\begin{align*}
&x_0 \quad \chi_0 \quad \gamma_0 \\
&(1, 1, 0, \leq) \quad (20, 1, -20, \leq) \quad (20, 1, -100, \leq) \\
&x_1 \quad x_2 \quad \chi_1 \quad \gamma_1 \\
&(1, 20, \infty, \leq) \quad (1, 1, 0, \leq) \quad (20, 20, -80, \leq) \\
&y_0 \quad (1, 20, \infty, \leq) \quad (20, 20, 80, \leq) \quad (1, 1, 0, \leq)
\end{align*}
$$

Furthermore, we intersect the set of states with the jump condition $\varphi$ to obtain $((M_0 \land inv(l_0))^0 \land inv(l_0) \land \varphi)$:

$$
\begin{align*}
&x_0 \quad \chi_0 \quad \gamma_0 \\
&(1, 1, 0, \leq) \quad (20, 1, -620, \leq) \quad (20, 1, -700, \leq) \\
&x_1 \quad x_2 \quad \chi_1 \quad \gamma_1 \\
&(1, 20, 920, \leq) \quad (1, 1, 0, \leq) \quad (20, 20, -80, \leq) \\
&y_0 \quad (1, 20, 1000, \leq) \quad (20, 20, 80, \leq) \quad (1, 1, 0, \leq)
\end{align*}
$$

(4) Finally, we reset the variables in set $\lambda = \{y\}$ to the corresponding value in set $\xi$. Here, $\xi_0 = 700$. So we obtain $M_1 = [\lambda \mapsto \xi](M_0 \land inv(l_0))^0 \land inv(l_0) \land \varphi)$, which is given by the matrix:

$$
\begin{align*}
&x_0 \quad \chi_0 \quad \gamma_0 \\
&(1, 1, 0, \leq) \quad (20, 1, -620, \leq) \quad (30, 1, -700, \leq) \\
&x_1 \quad x_2 \quad \chi_1 \quad \gamma_1 \\
&(1, 20, 920, \leq) \quad (1, 1, 0, \leq) \quad (30, 20, 13600, \leq) \\
&y_0 \quad (1, 30, 1000, \leq) \quad (20, 30, -4600, \leq) \quad (1, 1, 0, \leq)
\end{align*}
$$

Note that the last difference constraint matrix corresponds to the multirate zone:

$$(l_1, 620 \leq x ? \leq 920 \land 4600 \leq 30y ! - 20y ! \leq 13600 \land y ! = 700)
$$

Consequently, the successor state in the multirate zone automata is $s_1$. Repeating the same sequence of steps, we obtain the remaining states of the zone automata:

(1) $s_2 = (l_2, 820 \leq y ! \leq 940 \land 6600 \leq 30y ! - 30x ? \leq 10200 \land x ? = 600)$

(2) $s_3 = (l_3, 960 \leq x ? \leq 1080 \land -4800 \leq 20x ? - 30y ! \leq -2400 \land y ! = 800)$

(3) $s_4 = (l_4, 900 \leq y ! \leq 960 \land 0 \leq 20y ! - 20x ? \leq 1200 \land x ? = 900)$

(4) $s_5 = (l_5, 900 \leq x ? \leq 1000 \land 13000 \leq 30x ? - 20y ! \leq 16000 \land y ! = 700)$

(5) $s_6 = (l_6, 820 \leq y ! \leq 850 \land 6600 \leq 30y ! - 30x ? \leq 7500 \land x ? = 600)$
The reachability computation terminates at this point because the state $s_6$ is contained in $s_2$. Thus, no new states will be obtained by computing successor states in the zone automata.

### 6.3. MZIA Modeling

Now we model the above boiler system based on our model MZIA $P = (S_p, S'_p, A_p, \Delta_p, A_p^I, X_p, V_p, V_0^I, V_0^P, C_p, F_p, F_0^P, I_p, T_p)$ which consists of the following elements:

1. $S_p = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$;
2. $S'_p = \{s_0\}$;
3. $A_p = \{a_0, a_1, a_2, a_3\}$;
4. $V_p = \{?; !; \}$
5. Here we introduce a global clock variable $\text{clock} \in C_p$ and map all states to their corresponding Z schema by function $F_p^S$. For the sake of space, we only use state $s_0$ and state $s_1$ as an example, and the rest states are similar:

\[
F_p^S(s_0) = [I: \{l_0, l_1, l_2, l_3\}; x?: !; \text{clock} \leq 0];
\]

\[
F_p^S(s_1) = [I: \{l_0, l_1, l_2, l_3\}; x?: !; \text{clock} \leq 0];
\]

\[
1; l_0; 620 \leq x?; 920; 4600 \leq 30 x?; 20 y!; 13600; 30 \leq \text{clock} \leq 45];
\]

6. Also, we map all actions to their corresponding Z schema by function $F_p^A$ with an example of action $a_0$:

\[
F_p^A(a_0) = [y!: !];
\]

7. $T_p = \{(s_0, a_0, s_0), (s_1, a_1, s_1), (s_2, a_2, s_2), (s_3, a_3, s_3), (s_4, a_4, s_4), (s_5, a_5, s_5)\}$.

### 6.4. Verification

Next we illustrate the procedure for the approximated model checking on MZIAs by verifying several simple properties of the boiler system.

1. The pressure in the boiler will rise to at least 700(Kpa) after 30 clock cycles, which is represented by RT-DCL formula $f_1 = A(\text{true}_{\geq 30}, y! \geq 700)$. So we have $\text{Sub}(f_1) = (\text{true}, \text{clock} \geq 30, y! \geq 700)$. By our model checking algorithm, we label the state $s_1, s_2, s_3, s_4, s_5$ and $s_6$ with formula $\text{true}$, clock $\geq 30$ and $y! \geq 700$ in sequence. We allow the measuring errors of real-numbered variable $y!$ and suppose that the error $\delta = 0.2$. At this point, it is enough that the variable $y!$ is greater than or equal to 700 even if its actual measured value has a certain error range of 0.2. For example, we will still label state $s_1$ with formula $y! \geq 700$ even when the actual measured value of pressure in $s_1$ is 699.8. Next, all the states will be labeled with formula $A(\text{true}_{\geq 30}, y! \geq 700)$, including $s_0$. So the system satisfies the property above described by formula $A(\text{true}_{\geq 30}, y! \geq 700)$.

2. There exists some state that the temperature in the boiler remains between 900(°C) and 1000(°C), and the relationship between temperature and pressure satisfies formula $13000 \leq 30 x? - 20 y! \leq 16000$, which is represented by RT-DCL formula $f_2 = E(\text{true}\ U_{\leq 30}(x?), y! \leq 16000\ U_{\leq 30}(x?), y! \geq 13000)$.
(900 ≤ x? ≤ 1000 ∧ 13000 ≤ 30x? − 20y! ≤ 16000)) \). So we have \( \text{Sub}(f_2) = \{\text{true}, 900 ≤ x? ≤ 1000, 13000 ≤ 30x? − 20y! ≤ 16000\} \). As in the above model checking process, we label the state \( s_5 \) which satisfy the formulas \( \text{true} \), 900 ≤ x? ≤ 1000 and 13000 ≤ 30x? − 20y! ≤ 16000. Here, we allow the measuring error \( \delta = 0.5 \). If the relationship 30x? − 20y! in state \( s_5 \) is greater than or equal to 13000 but less than or equal to 16000.5, we still label \( s_5 \) with formula 13000 ≤ 30x? − 20y! ≤ 16000. At last, by the model checking algorithm, we find one computation \( \pi = (s_0, s_1, \ldots, s_5) \in \Gamma(s_0) \) such that for each \( 0 ≤ i < 5 \), \( s_i \in \{s \mid \text{true} \in \text{label}(s)\} \), \( s_5 \in \{s \mid 13000 ≤ 30x? − 20y! ≤ 16000 \in \text{label}(s)\} \). So the system satisfies \( E(\text{true} U(900 ≤ x? ≤ 1000 ∧ 13000 ≤ 30x? − 20y! ≤ 16000)) \).

7. Conclusion

In order to verify the “hybrid control” systems, we define a combination of interface automata, multirate hybrid automata and Z called MZIA, which can be applied to specify the behavior and data structures properties of a hybrid system. Moreover, it is intuitive, and it is easy to be understood and to be applied by programmers. In [6, 7], we proposed the ZIA and HZIA model, but didn’t give the corresponding temporal logic and the model checking algorithm. In [8], the author proposed the model checking procedure for the initialized multirate hybrid automata, but his model can describe the behavior of the interfaces between components and the data structures properties of the system. Furthermore, compared with current model checking techniques, we consider the measuring errors of real-numbered variables in practice and study the approximated model checking for MZIAs. Our work has great significance on the analyzing and verifying the control systems.

Acknowledgements

This paper was supported by the Aviation Science Fund of China under Grant No. 20128052064, the Fundamental Research Funds for the Central Universities under Grant No.NZ2013306, the National Basic Research Program of China (973 Program) under Grant No. 2014CB744903 and the National Natural Science Foundation of China under Grant No. 61272083.

References


Authors

Guozheng Li, He obtained his Bachelor degree in Computer Science Department of Nanjing University of Aeronautics & Astronautics and is currently a postgraduate in Computer Science Department of Nanjing University of Aeronautics & Astronautics. His research interests include formal methods and model checking.

Zining Cao, He is a professor in Computer Science Department of Nanjing University of Aeronautics & Astronautics. His work focuses on specification and verification for software systems. He obtained Bachelor degree and Master degree in 1995 and 1998, respectively, in Computer Science Department from Nanjing University of Aeronautics & Astronautics. He obtained Ph.D degree in 2001 in Computer Science Department from Tsinghua University.

Zheng Gao, He obtained his Bachelor degree in Computer Science Department of Nanjing Audit University and is currently a postgraduate in Computer Science Department of Nanjing University of Aeronautics & Astronautics. His research interests include formal methods.