International Journal of Control and Automation
Vol.9, No.2 (2016), pp.271-286
http://dx.doi.org/10.14257/ijca.2016.9.2.26

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

ISSN: 2005-4297 IJCA

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 $X=\{x_1,...,x_n\}$ 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 $x_i \sim c$, where $c\in\mathbb{Q}$, $x_i\in X$, and $\sim\in\{<,\leq,>,\geq\}$. We use B_i to denote its projection onto the ith coordinate, and $\Re(X)$ 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) $P=\langle S_P, S_P^i, A_P^I, A_P^O, A_P^H, X_P, V_P^I, V_P^O, V_P^H, C_P, F_P^S, F_P^A, I_P, T_P\rangle$ consists of the following elements:

(1) S_P is a set of states;

(2) $S_P^i \subseteq S_P$ is a set of initial states. If $S_P^i=\varnothing$ then P is called empty;

(3) A_P^I、A_P^O and A_P^H are disjoint sets of input, output, and internal actions, respectively. We denote by $A_P=A_P^I\cup A_P^O\cup A_P^H$ the set of all actions;

(4) $X_P=\{x_1,...,x_n\}$ is a finite set of real-numbered variables; The number n is called the dimension of P. We write $\dot{X}_P=\{\dot{x}_1,...,\dot{x}_n\}$ representing first derivatives during continuous change, and $X_P'=\{x_1',...,x_n'\}$ representing values at the conclusion of discrete change. $\Re(X_P)$ is a rectangle over X_P;

(5) V_P^I, V_P^O and V_P^H are disjoint sets of input, output, and internal variables, respectively. We denote by $V_P=V_P^I\cup V_P^O\cup V_P^H$ 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^I to an input operation schema $\Phi(V_P)$ in Z language, and maps any output action in A_P^O to an output operation

schema $\Phi(V_P)$ in Z language, and maps any internal action in A_P^H 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 $\mathbb{R}^n$ or $\Re(X_P)$, where $init_P : S_P \to \mathbb{R}^n$ assigns an initial condition to each state, $inv_P : S_P \to \Re(X_P)$ assigns an invariant condition to each state, and $act_P : S_P \to \mathbb{R}^n$ 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) $T_P \subseteq S_P \times A_P \times \Re(X_P) \times 2^{X_P} \times \mathbb{R}^n \times S_P$ 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 φ that specifies when the transition is enabled, and a set of real-numbered variables $\lambda \subseteq 2^{X_P}$ that are reset to the corresponding value in ξ when the transition is executed. In this paper, we define that if for every coordinate $i \in \{1,\ldots,n\}$ with $act_P(s)_i \neq act_P(s')_i$, then $x_i \in \lambda$. Furthermore, we have that $\models ((F_P^S(s) \wedge F_P^A(a)) \setminus (x_1, \cdots, x_m)$

(11) $\Leftrightarrow F_P^S(t)[y_1'/y_1, \cdots, y_n'/y_n])$, where $\{x_1, ..., x_m\}$ is the set of variables in $F_P^S(s)$, $\{y_1, ..., y_n\}$ is the set of the variables in $F_P^S(t)$, the set variables in $F_P^A(a)$ is the subset of $\{x_1, ..., x_m\} \cup \{y_1, \ldots, y_n\}$.

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 ϕ 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 $\{x_{i_1},\ldots,x_{i_m}\} \subseteq \{x_1,\ldots x_n\}$, where $\{x_{i_1},\ldots,x_{i_m}\}$ is the real-numbered variables set;

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

(3) If $\phi_1, \phi_2 \in$ *RT-DCL*, then $\phi_1 \vee \phi_2 \in$ *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 $\mathrm{EX}\phi \in$ *RT-DCL*;

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

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

(9) If ϕ_1, $\phi_2 \in$ *RT-DCL*, then $\mathrm{E}\phi_1 \mathrm{U}_{\sim c} \phi_2 \in$ *RT-DCL*, where $\sim \in \{<, \leq, =, \neq, >, \geq\}$ and $c \in \mathbb{R}$;

(10) If ϕ_1, $\phi_2 \in$ *RT-DCL*, then $\mathrm{A}\phi_1 \mathrm{U}_{\sim c} \phi_2 \in$ *RT-DCL*, where $\sim \in \{<, \leq, =, \neq, >, \geq\}$ and $c \in \mathbb{R}$.

(11) If $\phi \in$ *RT-DCL*, then $\mathrm{EG}_{\sim c}\phi \in$ *RT-DCL*, where $\sim \in \{<, \leq, =, \neq, >, \geq\}$ and $c \in \mathbb{R}$;

(12) If $\phi \in$ *RT-DCL*, then $AG_{\sim c}\phi \in$ *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 δ on $\{x_1,\ldots,x_m\}$ which represents the measuring errors of real-numbered variables $\{x_1,\ldots,x_m\}$, an assignment ρ on $\{y_1,\ldots,y_n\}$, where $\{x_1,\ldots,x_m\}$ are set of all real-numbered variables in $\{y_1,\ldots,y_n\}$. We use the notation $\rho \oplus \delta$ to denote the set of assignment $\{\sigma \mid \sigma(y) = \rho(y) \text{if } y \notin \{x_1,\ldots,x_m\}$, and $\sigma(x) = \rho(x) + a, \text{if } x \in \{x_1,\ldots,x_m\}, \text{where} -\delta(x) \leq a \leq \delta(x)\}$.

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 a_i, such that $(s_i, a_i, s_{i+1}) \in T_P$ for each $i \in \mathbb{N}$, i.e., s_i+1 is the result of performing an a_i on the state s_i. For a computation $\pi = (s_0, s_1, \cdots)$, let $\pi[k] = s_k$, and $\pi_k = (s_0, \cdots, s_k)$ 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 ϕ, and $s \in S_P$, we define the satisfaction relation $(P,s) \mid\approx \phi$ inductively:

(1) $(P,s) \mid\approx p(x_1,\ldots,x_n)$ iff $F_P^V(s) \mid\approx p(x_1,\ldots,x_n)$. We use relation $\mid\approx$ representing that $F_P^V(s)$ in Z schema approximately satisfy the formula $p(x_1,\ldots,x_n)$. Specifically, there exists an assignment ρ on $AV(F_P^V(s))$ and an positive real-numbered assignment δ on $CV(F_P^V(s))$ such that $\rho \oplus \delta \mid= p(x_1,\ldots,x_n)$, where $\rho \oplus \delta \mid= p(x_1,\ldots,x_n)$ means that there exists an assignment $\lambda \in \rho \oplus \delta$ with $p(x_1,\ldots,x_n)$ true;

(2) $(P,s) \mid\approx \phi_1 \wedge \phi_2$ iff $(P,s) \mid\approx \phi_1$ and $(P,s) \mid\approx \phi_2$;

(3) $(P,s) \mid\approx \phi_1 \vee \phi_2$ iff $(P,s) \mid\approx \phi_1$ or $(P,s) \mid\approx \phi_2$;

(4) $(P,s) \mid\approx (\forall x:T)\phi$ iff $(P,s) \mid\approx \bigcap_{v \in T} \phi\{v/x\}$;

(5) $(P,s) \mid\approx (\exists x:T)\phi$ iff $(P,s) \mid\approx \bigcup_{v \in T} \phi\{v/x\}$;

(6) $(P,s) \mid\approx \mathrm{EX}\phi$ iff there exists a state $s' \in S_P$ such that $(s,a,s') \in T_P$ and $(P,s') \mid\approx \phi$;

(7) $(P,s) \mid\approx \mathrm{AX}\phi$ iff for each state $s' \in S_P$ and $(s,a,s') \in T_P$, we have $(P,s') \mid\approx \phi$;

(8) $(P,s) \mid\approx c\square\phi$ iff $(P,s)_{c\mapsto 0} \mid\approx \phi$, where $(P,s)_{c\mapsto 0}$ means that the clock variable is set to zero, and $(P,s)_{c\mapsto 0} \mid\approx \phi$ means that ϕ is still true in state (P,s) after *c* is reset to zero;

(9) $(P,s) \mid\approx \mathrm{E}\phi_1 \mathrm{U}_{\sim c} \phi_2$ iff there exists a computation $\pi \in \prod(s)$ such that $\pi[t] \mid\approx \phi_2$, where $t \sim c$, and $\pi[t'] \mid\approx \phi_1$ for each $t' \in (0,t)$;

(10) $(P,s)\mid\approx \mathrm{A}\phi_1 \mathrm{U}_{\sim c}\phi_2$ iff for each computation $\pi \in \prod(s)$, we have $\pi[t]\mid\approx \phi_2$, where $t \sim c$, and $\pi[t']\mid\approx \phi_1$ for each $t' \in (0,t)$.

(11) $(P,s)\mid\approx \mathrm{EG}_{\sim c}\phi$ iff there exists a computation $\pi \in \prod(s)$ such that $\pi[t]\mid\approx \phi$, where $t \sim c$;

(12) $(P,s)\mid\approx \mathrm{AG}_{\sim c}\phi$ iff for each computation $\pi \in \prod(s)$, we have $\pi[t]\mid\approx \phi$, where $t \sim c$.

4. MZIA with Finite Domain

Consider a MZIA P and a pair $(s, D_P) \in S_P \times$ □, where □ 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: $ax - by \prec c$, $x \prec c$, and $c \prec x$, where $\prec \in \{<, \leq\}$, $c \in$ □. 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 s □ $[v_1 : T_1; \ldots; v_m : T_m \mid P_1; \ldots; P_n]$, 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 $P = \left\langle S_P, S_P^i, A_P^I, A_P^O, A_P^H, X_P, V_P^I, V_P^O, V_P^H, C_P, F_P^S, F_P^A, I_P, T_P \right\rangle$ 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 $\mathrm{F}(P)$. Then, we label the zones in $\mathrm{F}(P)$ with subformulas of *f*, starting from the subformulas of length 1, then of length 2, and so on. If the initial states of $\mathrm{F}(P)$ are labeled with *f*, then $\mathrm{F}(P)$ 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 $\mathrm{F}(P)$. Next, approximated model checking for MZIA with finite domain $\mathrm{F}(P)$ 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 $(\mathrm{F}(P), s) \approx f$ iff $f \in label(s)$.

We define the function *Sub*, when given a formula ϕ, returns a queue of syntactic subformulas of ϕ such that if ϕ_1 is a subformula of ϕ and ϕ_2 is a subformula of ϕ_1, then ϕ_2 precedes ϕ_1 in the queue $Sub(\phi)$. Procedure for labeling the states of $\mathrm{F}(P)$ satisfying some *RT-DCL* formula *f* is as follows:

For each Ψ in *Sub* (*f*) do:
case $\Psi = p(x_1,...,x_n)$:
if $F_P^V(s) \mid\approx p(x_1,...,x_n)$ then
label(*s*)= *label*(*s*)$\cup\Psi$;
break; //where Z schema $F_P^V(s)$ is regarded as a first order logical formula
case $\Psi=\phi_1\wedge\phi_2$:
if $\phi_1\in$*label*(*s*) and $\phi_2\in$*label*(*s*) then
label(*s*)=*label*(*s*)$\cup\Psi$;
break;
case $\Psi=\phi_1\vee\phi_2$:
if $\phi_1\in$*label*(*s*) or $\phi_2\in$*label*(*s*) then
label(*s*)=*label*(*s*)$\cup\Psi$;
break;
case $\Psi=(\forall x{:}T)\phi$:
if $s\in\bigcap_{u\in T}\{t\mid\phi\{u/x\}\in label(t)\}$ then
label(*s*)= *label*(*s*)$\cup\Psi$;
break;
case $\Psi=(\exists x{:}T)\phi$:
if $s\in\bigcup_{u\in T}\{t\mid\phi\{u/x\}\in label(t)\}$ then
label(*s*)= *label*(*s*)$\cup\Psi$;
break;
case $\Psi=$ EXϕ:
if for some state $s'\in S_P$ and $(s,a,s')\in T_P$ such that $s'\in\{t\mid\phi\in label(t)\}$, then
label(*s*)= *label*(*s*)$\cup\Psi$;
break;
case $\Psi=$ AXϕ:
if for each state $s'\in S_P$ and $(s,a,s')\in T_P$ such that $s'\in\{t\mid\phi\in label(t)\}$, then
label(*s*)= *label*(*s*)$\cup\Psi$;
break;
case $\Psi=c\Box\phi$:
if $s\in\{s\mid c=0$ *and* $\phi\in label(s)\}$ then
label(*s*)= *label*(*s*)$\cup\Psi$; // where *c* is a clock variable
break;
case $\Psi=$E(ϕ_1U$_{\sim c}\phi_2$):
if for some $\pi=(s_1,s_2,\ldots,s_n)\in\prod(s)$ such that for each $1\le i<n$, $s_i\in\{t\mid\phi_1\in label(t)\}$, $s_n\in\{t\mid\phi_2\in label(t)\}$ and $z\sim c$, then
label(*s*)= *label*(*s*)$\cup\Psi$;
// where *z* is a clock variable
break;
case $\Psi=$A(ϕ_1U$_{\sim c}\phi_2$):
if for all $\pi=(s_1,s_2,\ldots,s_n)\in\prod(s)$ such that for each $1\le i<n$, $s_i\in\{t\mid\phi_1\in label(t)\}$, $s_n\in\{t\mid\phi_2\in label(t)\}$ and $z\sim c$, then
label(*s*)= *label*(*s*)$\cup\Psi$;
break;
case $\Psi=$ EG$_{\sim c}\phi$:
if for some $\pi=(s_1,s_2,\ldots,s_n)\in\prod(s)$ such that for each $1\le i\le n$, $s_i\in\{t\mid\phi\in label(t)\}$, and $z\sim c$, then
label(*s*)= *label*(*s*)$\cup\Psi$;
break;
case $\Psi=$AG$_{\sim c}\phi$:
if for all $\pi=(s_1,s_2,\ldots,s_n)\in\prod(s)$ such that for each $1\le i\le n$, $s_i\in\{t\mid\phi\in label(t)\}$, and $z\sim c$, then
label(*s*)= *label*(*s*)$\cup\Psi$;
break;

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 Ψ. Assume that *s* is labeled with Ψ iff $(\mathrm{F}(P),s)\mid\approx\Psi$.

(1) If Ψ is atomic proposition formula $p(x_1,\ldots,x_n)$, and *s* is labeled with Ψ, then $F_P^V(s)\mid\approx p(x_1,...,x_n)$. So we have $(\mathrm{F}(P),s)\mid\approx\Psi$;

(2) If $\Psi=\phi_1\wedge\phi_2$, and *s* is labeled with Ψ, then *s* is also labeled with ϕ_1 and ϕ_2. By the induction hypothesis, $(\mathrm{F}(P),s)\mid\approx\phi_1$ and $(\mathrm{F}(P),s)\mid\approx\phi_2$. So we have $(\mathrm{F}(P),s)\mid\approx\phi_1\wedge\phi_2$, i.e., $(\mathrm{F}(P),s)\mid\approx\Psi$;

(3) If $\Psi = \phi_1 \vee \phi_2$, and s is labeled with Ψ, then s is labeled with either ϕ_1 or ϕ_2. By the induction hypothesis, $(\mathrm{F}(P), s) \approx \phi_1$ or $(\mathrm{F}(P), s) \approx \phi_2$. So we have $(\mathrm{F}(P), s) \approx \phi_1 \vee \phi_2$, i.e., $(\mathrm{F}(P), s) \approx \Psi$;

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

(5) If $\Psi = (\exists x : T)\phi$, and s is labeled with Ψ, then for some $u \in T$, s is labeled with $\phi\{u / x\}$. By the induction hypothesis, we have that $(\mathrm{F}(P), s) \approx \phi\{u/x\}$, for some $u \in T$, i.e., $(\mathrm{F}(P), s) \approx \Psi$;

(6) If $\Psi = \mathrm{EX}\phi$, and s is labeled with Ψ, then there exists some state $s' \in S_P$ and $(s, a, s') \in T_P$ such that s' is labeled with ϕ. By the induction hypothesis, we have that $(\mathrm{F}(P), s) \approx \mathrm{EX}\phi$, i.e., $(\mathrm{F}(P), s) \approx \Psi$;

(7) If $\Psi = \mathrm{AX}\phi$, and s is labeled with Ψ, then for each state $s' \in S_P$ and $(s, a, s') \in T_P$, we have s' is labeled with ϕ. By the induction hypothesis, we have that $(\mathrm{F}(P), s) \approx \mathrm{AX}\phi$, i.e., $(\mathrm{F}(P), s) \approx \Psi$;

(8) If $\Psi = c\phi_1$, and s is labeled with Ψ, then s is still labeled with ϕ_1 after the clock variable c is reset to 0. By the induction hypothesis, $(\mathrm{F}(P), s)_{c\mapsto 0} \approx \phi$. So we have $(\mathrm{F}(P), s) \approx \Psi$;

(9) If Ψ is $\mathrm{E}\phi_1 \mathrm{U}_{\sim c}\phi_2$, and s is labeled with Ψ, then there exists a computation $\pi = (s_1, s_2, \ldots, s_n) \in \prod(s)$ such that s_i is labeled with ϕ_1 for each $1 \le i < n$, s_n is labeled with ϕ_2, and clock variable z satisfy $z \sim c$. By the induction hypothesis, there exists a computation $\pi = (s_1, s_2, \ldots, s_n) \in \prod(s)$ such that $(\mathrm{F}(P), s_i) \approx \phi_1$, $(\mathrm{F}(P), s_n) \approx \phi_2$ and $z \sim c$. So we have $(\mathrm{F}(P), s) \approx \mathrm{E}(\phi_1 \mathrm{U}_{\sim c}\phi_2)$, i.e., $(\mathrm{F}(P), s) \approx \Psi$.

(10) If Ψ is $\mathrm{A}\phi_1 \mathrm{U}_{\sim c}\phi_2$, and s is labeled with Ψ, then every computation $\pi = (s_1, s_2, \ldots, s_n) \in \prod(s)$ satisfies that s_i is labeled with ϕ_1 for each $1 \le i < n$, s_n is labeled with ϕ_2, and clock variable z satisfy $z \sim c$. By the induction hypothesis, every computation $\pi = (s_1, s_2, \ldots, s_n) \in \prod(s)$ such that $(\mathrm{F}(P), s_i) \approx \phi_1$, $(\mathrm{F}(P), s_n) \approx \phi_2$ and $z \sim c$. So we have $(\mathrm{F}(P), s) \approx \mathrm{A}(\phi_1 \mathrm{U}_{\sim c}\phi_2)$, i.e., $(\mathrm{F}(P), s) \approx \Psi$.

(11) If Ψ is $\mathrm{EG}_{\sim c}\phi$, and s is labeled with Ψ, then there exists a computation $\pi = (s_1, s_2, \ldots, s_n) \in \prod(s)$ such that s_i is labeled with ϕ for each $1 \le i \le n$, and clock variable z satisfy $z \sim c$. By the induction hypothesis, there exists a computation $\pi = (s_1, s_2, \ldots, s_n) \in \prod(s)$ such that $(\mathrm{F}(P), s_i) \approx \phi$ and $z \sim c$. So we have $(\mathrm{F}(P), s) \approx \mathrm{EG}_{\sim c}\phi$, i.e., $(\mathrm{F}(P), s) \approx \Psi$.

(12) If Ψ is $\mathrm{AG}_{\sim c}\phi$, and s is labeled with Ψ, then every computation $\pi = (s_1, s_2, \ldots, s_n) \in \prod(s)$ satisfies that s_i is labeled with ϕ for each $1 \le i \le n$, and clock variable z satisfy $z \sim c$. By the induction hypothesis, every computation $\pi = (s_1, s_2, \ldots, s_n) \in \prod(s)$ satisfies $(\mathrm{F}(P), s_i) \approx \phi$, and $z \sim c$. So we have $(\mathrm{F}(P), s) \approx \mathrm{AG}_{\sim c}\phi$, i.e., $(\mathrm{F}(P), s) \approx \Psi$.

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;
        display();
};                          //formula in the Z schema
```

(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.

```
struct sys_node{                    //nodes of system
        state s;                    // state
        vector<edge_node> e;        //edge
};
```

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

```
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 $succ(M, e)$ 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.

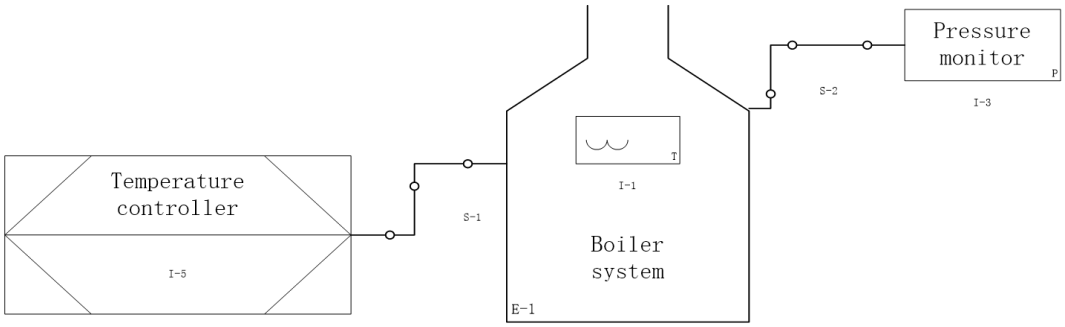

Figure 1. Boiler Plant

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 Respresentation

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].

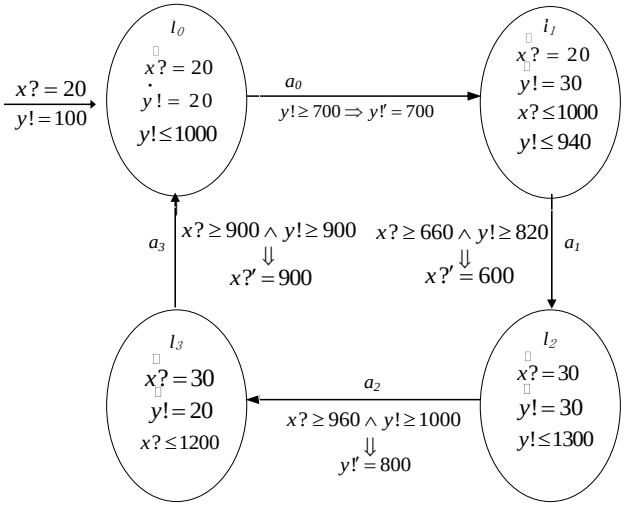

Figure 2. States Transitions of Boiler System

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

	x_0	$x?$	$y!$
x_0	(1, 1, 0, ≤)	(20, 1, -20, ≤)	(20, 1, -100, ≤)
$x?$	(1, 20, 20, ≤)	(1, 1, 0, ≤)	(20, 20, -80, ≤)
$y!$	(1, 20, 100, ≤)	(20, 20, 80, ≤)	(1, 1, 0, ≤)

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 $inv(l_0) = y! \le 1000$, which is given by the matrix:

	x_0	$x?$	$y!$
x_0	(1, 1, 0, ≤)	(20, 1, ∞, ≤)	(20, 1, ∞, ≤)
$x?$	(1, 20, ∞, ≤)	(1, 1, 0, ≤)	(20, 20, ∞, ≤)
$y!$	(1, 20, 1000, ≤)	(20, 20, ∞, ≤)	(1, 1, 0, ≤)

(2) Next, we let time elapse in the location l_0 using the operator $\Uparrow$. The matrix for $(\mathrm{M}_0 \wedge inv(l_0))^{\Uparrow}$ is:

	x_0	$x?$	$y!$
x_0	(1, 1, 0, ≤)	(20, 1, -20, ≤)	(20, 1, -100, ≤)
$x?$	(1, 20, ∞, ≤)	(1, 1, 0, ≤)	(20, 20, -80, ≤)
$y!$	(1, 20, ∞, ≤)	(20, 20, 80, ≤)	(1, 1, 0, ≤)

(3) The jump condition $\varphi = 700 \le y!$ for the a_0 transition from location l_0 to location l_1 is:

	x_0	$x?$	$y!$
x_0	(1, 1, 0, ≤)	(20, 1, ∞, ≤)	(20, 1, -700, ≤)
$x?$	(1, 20, ∞, ≤)	(1, 1, 0, ≤)	(20, 20, ∞, ≤)
$y!$	(1, 20, ∞, ≤)	(20, 20, ∞, ≤)	(1, 1, 0, ≤)

Furthermore, we intersect the set of states with the jump condition φ to obtain $((\mathrm{M}_0 \wedge inv(l_0))^{\Uparrow} \wedge inv(l_0) \wedge \varphi)$:

	x_0	$x?$	$y!$
x_0	(1, 1, 0, ≤)	(20, 1,-620, ≤)	(20, 1, -700, ≤)
$x?$	(1, 20, 920, ≤)	(1, 1, 0, ≤)	(20, 20, -80, ≤)
$y!$	(1, 20,1000, ≤)	(20, 20, 80, ≤)	(1, 1, 0, ≤)

(4) Finally, we reset the variables in set $\lambda = \{y!\}$ to the corresponding value in set ξ. Here, $\xi_{y!} = 700$. So we obtain $\mathrm{M}_1 = [\lambda \mapsto \xi]((\mathrm{M}_0 \wedge inv(l_0))^{\Uparrow} \wedge inv(l_0) \wedge \varphi)$, which is given by the matrix:

	x_0	$x?$	$y!$
x_0	(1, 1, 0, ≤)	(20, 1, -620, ≤)	(30, 1, -700, ≤)
$x?$	(1,20,920, ≤)	(1, 1, 0, ≤)	(30,20,13600, ≤)
$y!$	(1,30,1000, ≤)	(20,30,-4600, ≤)	(1, 1, 0, ≤)

Note that the last difference constraint matrix corresponds to the multirate zone:
$s_1 = (l_1, 620 \le x? \le 920 \wedge 4600 \le 30x? - 20y! \le 13600 \wedge y! = 700)$

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

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

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

(3) $s_4 = (l_0, 900 \le y! \le 960 \wedge 0 \le 20y! - 20x? \le 1200 \wedge x? = 900)$

(4) $s_5 = (l_1, 900 \le x? \le 1000 \wedge 13000 \le 30x? - 20y! \le 16000 \wedge y! = 700)$

(5) $s_6 = (l_2, 820 \le y! \le 850 \wedge 6600 \le 30y! - 30x? \le 7500 \wedge x? = 600)$

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 $P = \langle S_P, S_P^i, A_P^I, A_P^O, A_P^H, X_P, V_P^I, V_P^O, V_P^H, C_P, F_P^S, F_P^A, I_P, T_P \rangle$ which consists of the following elements：

(1) $S_P = \{s_0, s_1, s_1, s_2, s_3, s_4, s_5, s_6\}$;

(2) $S_P^i = \{s_0\}$;

(3) $A_P = \{a_0, a_1, a_2, a_3\}$;

(4) $V_P = \{x?; y!; l\}$;

(5) Here we introduce a global clock variable $clock \in C_P$ 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: $F_P^S(s_0) \triangleq [l : \{l_0, l_1, l_2, l_3\}; x? : \square; y! : \square, clock : C_P \mid l = l_0; x? = 20; y! = 100; clock = 0]$;
$F_P^S(s_1) \triangleq [l : \{l_0, l_1, l_2, l_3\}; x? : \square; y! : \square, clock : C_P \mid l = l_1; 620 \le x? \le 920; 4600 \le 30x? - 20y! \le 13600; y! = 700; 30 \le clock \le 45]$;

(6) Also, we map all actions to their corresponding Z schema by function F_P^A with an example of action a0: $F_P^S(a_0) \triangleq [y! : \square \mid y! = 700]$;

(7) $T_P = \{(s_0, a_0, s_1), (s_1, a_1, s_2), (s_2, a_2, s_3), (s_3, a_3, s_4), (s_4, a_0, s_5), (s_5, a_1, s_6)\}$.

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 $f_1 = A(true \mathrm{U}_{\ge 30} y! \ge 700)$. So we have $Sub(f_1) = \{true, clock \ge 30, y! \ge 700\}$. By our model checking algorithm, we label the state s_1, s_2, s_3, s_4, s_5 and s_6 with formula $true$, $clock \ge 30$ and $y! \ge 700$ 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 $y! \ge 700$ even when the actual measured value of pressure in s_1 is 699.8 . Next, all the states will be labeled with formula $A(true \mathrm{U}_{\ge 30} y! \ge 700)$, including s_0. So the system satisfies the property above described by formula $A(true \mathrm{U}_{\ge 30} y! \ge 700)$.

(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 $13000 \le 30x? - 20y! \le 16000$, which is represented by *RT-DCL* formula $f_2 = E(true \mathrm{U}$

$(900 \le x? \le 1000 \wedge 13000 \le 30x? - 20y! \le 16000))$. So we have $Sub(f_2) = \{true, 900 \le x? \le 1000,\ 13000 \le 30x? - 20y! \le 16000\}$. As in the above model checking process, we label the state s_5 which satisfy the formulas $true$, $900 \le x? \le 1000$ and $13000 \le 30x? - 20y! \le 16000$. 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 $13000 \le 30x? - 20y! \le 16000$. At last, by the model checking algorithm, we find one computation $\pi = (s_0, s_1, \ldots, s_5) \in \prod(s_0)$ such that for each $0 \le i < 5$, $s_i \in \{s \mid true \in label(s)\}$, $s_5 \in \{s \mid 13000 \le 30x? - 20y! \le 16000 \in label(s)\}$. So the system satisfies $E(true\mathrm{U}(900 \le x? \le 1000 \wedge 13000 \le 30x? - 20y! \le 16000))$.

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't 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

[1] L.D. Alfaro and T.A. Henzinger, "Interface Automata", Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering Acm, **(2001)**, pp. 109-120.
[2] J.M. Spivey, "The Z notation: a reference manual", Prentice Hall International (UK) Ltd., **(1992).**
[3] R.M.Z. Guang-Quan, "New Approaches for Model Checking", Computer Science, vol. 5, **(2003)**, p. 028.
[4] B. Bérard, M. Bidoit and A. Finkel, "Systems and software verification: model-checking techniques and tools [M]", Springer Publishing Company, Incorporated, **(2010)**.
[5] Z. Cao and H. Wang, "Fundamentals of Software Engineering", Springer Berlin Heidelberg, **(2012).**
[6] Z. Cao and H. Wang , "Hybrid ZIA and its Approximated Refinement Relation", In proceeding of: ENASE 2011 -Proceedings of the 6th International Conference on Evaluation of Novel Approaches to Software Engineering, Beijing, China, 8-11 June, **(2011)**, pp. 260-265.
[7] H. Zhang, "Model Checking Multirate Hybrid Systems with Restricted Convex Polyhedron", //Sixth International Symposium on Theoretical Aspects of Software Engineering, IEEE, **(2011), pp.** 93-99.
[8] T.A. Henzinger, P.W. Kopke and A. Puri, "What's Decidable About Hybrid Automata?". Journal of Computer and System Sciences, vol. 57, **(1998)**, pp. 94-124.
[9] R. Alur, C. Courcoubetis and D. Dill, "Model-checking for real-time systems", //IEEE Symposium on Logic in Computer Science, Lics. IEEE, **(1990)**, pp. 414-425.
[10] Z. Haibin and D. Zhenhua, "Symbolic Reachability Analysis of Multirate Hybrid Systems", Journal of Xi'an Jiaotong University, vol. 41, no. 4, **(2007)**, pp. 412-415.
[11] R. Alur, C. Courcoubetis and D. Dill, "Model-checking in dense real-time", Information and computation, vol. 104, no. 1, **(1993)**, pp. 2-34.

[12] K.G. Larsen, P. Pettersson and W. Yi, "Uppaal in a Nutshell", International Journal on Software Tools for Technology Transfer (STTT), vol. 1, no. 1, **(1997)**, pp. 134-152.
[13] B. lei, Y. Li and L. Wang, "BACH: A Toolset for Bounded Reachability Analysis of Linear Hybrid Systems ", Journal of Software, vol. 22, no. 4, **(2011)**, pp. 640-658.
[14] T.A. Henzinger and H. Wong-Toi, "Using HyTech to synthesize control parameters for a steam boiler", Springer Berlin Heidelberg, **(1996)**.

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.