Computers in Industry 62 (2011) 23–31

Contents lists available at ScienceDirect

Computers in Industry

journal homepage: www.elsevier.com/locate/compind

Formal modeling and synthesis of programmable logic controllers

Rui Wang [a,b,c,*], Xiaoyu Song [d], Jianzhong Zhu [a,b], Ming Gu [a,b]

[a] School of Software, Tsinghua University, Beijing, China
[b] Key Lab for ISS of MOE, Tsinghua University, Beijing, China
[c] Department of Computer Science, Tsinghua University, Beijing, China
[d] ECE Dept, Portland State University, Portland, OR, USA

ARTICLE INFO

Article history:
Received 11 June 2009
Received in revised form 28 February 2010
Accepted 31 May 2010
Available online 1 August 2010

Keywords:
PLC
Formal specification
Code synthesis
Embedded software

ABSTRACT

Programmable logic controllers (PLCs) are complex cyber-physical systems which are widely used in industry. This paper presents a robust approach to design and implement PLC-based embedded systems. Timed automata are used to model the controller and its environment. We validate the design model with resort to model checking techniques. We propose an algorithm to generate PLC code from timed automata and implement this algorithm with a prototype tool. This method can condense the developing process and guarantee the correctness of PLC programs. A case study demonstrates the effectiveness of the method.

1. Introduction

Programmable logic controllers (PLCs) are widely used in industry for safety critical embedded systems. The conventional PLC software development process is composed of requirement analysis, software and hardware design, software programming and testing. The testing for PLC costs a lot of time and money. Programs should be downloaded to PLC. Then connect all physical plants and do system testing. The fatal errors are usually caused by flimsy and changeful environments and logic errors in programs. Due to rapidly increasing system complexity, shortening time-to-market, and growing demand for real-time systems, formal methods are becoming indispensable in complementing the conventional testing methods.

We present a reliable design approach to modeling and synthesis of PLC systems. This paper focuses on automatical code generation from high level abstract models. As PLC system is real-time interactive system, we choose timed automata as the design model. Controllers and physical environments are modeled by timed automata. The design model is validated to satisfy the required safety and functional requirements with resort to a symbolic model checker Uppaal [1–3] for real-time systems. The model is improved until all relevant properties are satisfied. This means the design model behaves correctly under all circumstances. We have introduced this design process in [4]. We propose an algorithm to transfer design model to executable PLC code. The algorithm has been implemented in a prototype tool. Automatic code generation can reduce the dependence of testing. This method can find the errors of controllers at earlier time and simplify the developing process. A case study illustrates the proposed strategies.

Several formal techniques are used to model and analyze PLC-based embedded systems. Sacha used the finite state time machine (FSTM), an extension of Moore automata with time feature, to model PLC systems and generate codes automatically in [5,6]. However, the FSTM does not brace verification tools, we do not know whether the design model is correct or not. So an FSTM model has to be translated to timed automata of Uppaal for verification[7]. Frey developed a signal interpreted petri net (SIPN) editor for PLCs. This tool can generate PLC codes from SIPN model [8]. The notion of time is introduced into SIPN by timed places. Each timed place is associated with a minimal time constant, which is the time span a token has to spend in the place. Comparing to timed automata, this kind of time extension is less expressive. In [9], Pollmacher focuses on proving the correctness of translation from Time automata to ST (structured text) language. ST is a high level program language, while our target language is low level and relay based, so the translation method is different. He abstracts a correct model from generated code, and checks whether this model is a refinement of the design model.

The paper is organized as follows. Section 2 shows the background about PLC and timed automata. Section 3 presents formal specification and code generation. An example is used to show the design methodology. Section 4 concludes the paper.

* Corresponding author at: Room 1-118, FIT Building, Tsinghua University, Beijing 100084, China. Tel.: +86 10 62783549.
E-mail address: wang04@mails.tsinghua.edu.cn (R. Wang).

0166-3615/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.compind.2010.05.015

2. Background

2.1. PLC programs

A PLC controls several physical plants concurrently. It receives signals from sensors and produces control commands to actuators cyclicly. At the beginning of a scan cycle, the values of sensors are written to the input memory area. Then the PLC program is executed and the output value is written back to the memory. At the end of the scan cycle, the output memory is mapped to the actuators. A PLC embedded software system is different from conventional software. It deals with control signals for interaction of physical environments rather than data computation.

The International Electrotechnical Commission (IEC) published IEC61131 standard [10] for programmable logic controllers. Five program languages are defined by IEC. They are Instruction List (IL), Ladder Diagram (LD), Structured Text (ST), Function Block Diagram (FBD) and Sequential Function Chart (SFC). LD which is used most widely in industry is a relay diagram like language. We choose S7-300 LD program language used in Siemens PLCs [11] as our objective language.

LD program consists of graphical components of logic networks. Only one network is executed from left to right and then top to bottom at a time. A LD program emulates the flow of electric current through a series of logical input conditions and enable output conditions. A LD program includes a energized power rail in the left. Contacts represent logic input conditions such as switches, buttons or sensors. Closed contacts allow energy to flow through them to the next element. Normally open contact (-||-) is closed when the bit value stored above this contact equals to 1. Then ladder rail power flows across the contact. It is opposite for normally closed contact(-|/|-). Output coil -( ) is the logic output results or intermediate results. If there is power flow to the coil, the bit above is set to 1.

Reset coil -(R) is executed if the power flows to it. the address of the bit is reset to 0. The address can also be a timer. This operation will reset the time value to 0. It is opposite for set coil -(S). The two instructions together form a box of S-R flip flop, see Fig. 1. Boxes stand for additional instructions such as timers, integer mathematical instructions, comparison instructions and move instructions.

Comparison instruction *CMP?I* or *CMP?D* can be used as a normal contact. Symbol "?" can be = =, >=, <=, >, <, and < >. The comparison result is linked to RLO (Result of Logic) of the whole rung. Fig. 2 shows the LD instructions. *IN*1 and *IN*2 can be data of L, Q, I, M, D memory area or constant. Only when the box input condition is satisfied, the instruction is executed.

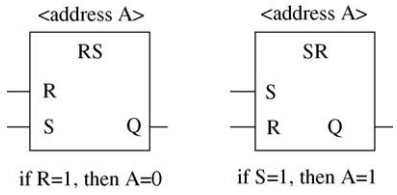

Fig. 1. The RS and SR instructions.

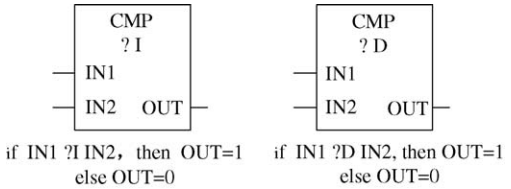

Fig. 2. The compare instructions.

Integer mathematical instructions shown in Fig. 3 include *SUB_I*, *ADD_I*, *MUL_I* and *DIV_I*. For example *ADD_I* is activated by a logic "1" at EN input. IN1 and IN2 are added and the result is sent to OUT. If the sum overflows, the ENO is logic "0".

2.2. Timed automata

Timed automata [12,13] are finite automata with extension of time variables for real-time systems. Here we use a variant of timed automata defined in [14]. Let V be a finite set of variables including clock variables C and data variables X, $V = C \cup X$ and $C \cap X = \Phi$. All clock variables advance simultaneously. We use $\psi(V)$ to denote invariant and guard formulas. We have $\psi := e|\psi \cap \psi$, e takes the form like $c \sim n$ or $x \sim n$, $c \in C, x \in X, \sim \in \{\leq, \geq, =, <, >\}$ and $n \in \mathbb{N}$. The assignment operation is defined as v:=*expresion*, $v \in V$. Let U denote all the assignment formulas.

Definition 1. Timed automaton is a tuple $\mathcal{A}(L, l_0, A, V, I, E)$

- L: a finite set of locations
- l_0: initial locations
- A: a finite set of actions
- V: a finite set of variables
- $I : L\psi(V)$ a location constraint function
- E: a finite set of edges, $E \subseteq L \times \psi \times A \times U \times L$

Each edge has a source location l, a target location l'. When guard $g \in \psi$ is satisfied, the transition happens and a subset of variables in V are updated by formula $r \in U$. An edge $e\ \langle l, g, a, r, l' \rangle$ can be written as $l \xrightarrow{g,a,r} l'$.

The automaton starts at the initial location l_0 with all clocks initialized 0. With time passing by, the clock variables increase at the same rate satisfying the invariant constraints $I(l_0)$. The system can remain at this location or transit to l_1 if the variables satisfy an edge enabling guard g. With the transition, action a is taken and variables are updated by formula r.

Definition 2. The semantics of a timed automaton $\mathcal{A}(L, l_0, A, V, I, E)$ is defined as a labeled transition system $\mathcal{S}(\mathcal{A}) = \langle S, s_0, \rightarrow \rangle$. $S \subseteq L \times U$ is a set of states, s_0 is the initial state, $\rightarrow \subseteq S \times (U \cup A) \times S \times S$ is the set of relations, divided into the following two classes.

- Elapses of time transitions: for $d \in \mathbb{R}^+$, $(l, u) \xrightarrow{d} (l, u + d)$, if for $\forall\, d' \leq d$, u and $u + d'$ satisfy $I(l)$, and
- Location switch transitions: $(l, u) \xrightarrow{a} (l', u')$, if $\exists\, e(l, a, g, r, l') \in E)$, $u' = r(u)$, u satisfies guard g, and u' satisfies $I(l')$.

A real-time system is composed of several components. The controller communicates with the physical environments in environment concurrently. So we employ timed automata networks $\overline{\mathcal{A}} = \mathcal{A}_1 || \ldots || \mathcal{A}_n$, and $\mathcal{A}_i = (L_i, l_i^0, A_i, V_i, I_i, E_i)$, n is the automata number in the network. These automata share a common set of action variables. Vector $\bar{l} = (l_1, \ldots, l_n)$ is the location vector of timed automata network $\overline{A}$. The invariant function $I(\bar{l})$ is the conjunctions of the constraints of all $\mathcal{A}_i$, $I(\bar{l}) = \wedge_i I_i(l_i)$. $\bar{l}[l_{i'}/l_i]$ denotes that l_i of $\bar{L}$ is replaced by $l_{i'}$.

Definition 3. The semantics of a timed automata network is defined as a transition system $S(\overline{A}) = \langle S, s_0, \rightarrow \rangle$, $S = (L_1 \times \ldots \times L_n) \times U$ is the set of states, s_0 is the initial state and $\rightarrow \subseteq S \times S$ is the transition relations defined as follows:

- $(\bar{l}, u) \xrightarrow{d} (\bar{l}, u + d)$, for $d \in \mathbb{R}^+$, if $\forall\, d' \leq d$, u and $u+d'$ satisfy $I(\bar{l})$,
- $(\bar{l}, u) \xrightarrow{a} (\bar{l}[l_{i'}/l_i], u')$, if there is an edge $e(l_i, a, g, r, l_{i'}) \in E$, $u' = r(u)$, u satisfies guard g and u' satisfies $I(\bar{l})$.

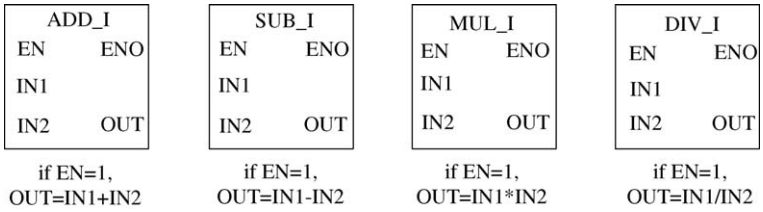

Fig. 3. The integer math instructions.

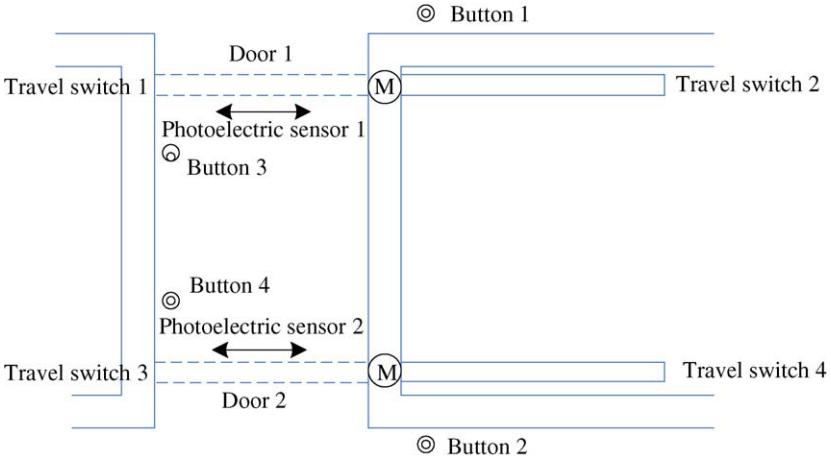

Fig. 4. The schema of double doors.

- $(\bar{l}, u) \xrightarrow{a} (\bar{l}[l_{j'}/l_j, l_{i'}/l_i], u')$, if there exits an edge $l_i \xrightarrow{a?, g_i, r_i} l_{i'}$, and another edge $l_j \xrightarrow{a!, g_j, r_j} l_{j'}$, $u' = r_i \cup r_j(u)$, u satisfies guard g, u' satisfies $I(\bar{l})$. a is the synchronization signal, $a?$ means receiving signal a, and $a!$ means sending signal a.

Transitions of different timed automata are synchronized by sending and receiving signals via channels.

3. Formal specification and code synthesis

We use an example to illustrate our iterative methodology.

3.1. An example: a double door controller

A special room needs to be protected from dust. So the entrance to the room is equipped with double doors. These two doors cannot be opened at the same time. The detail of this control system is shown in Fig. 4. There are buttons inside and outside the doors. Outside button *Button 1* and inside button *Button 3* can open the Door 1. There is a photoelectric sensor on Door 1. If a person stands inside Door 1 and block *Photoelectric sensor 1*, the first door can open automatically or keep opening. 3 s after the door opened completely, Door 1 closes automatically. While closing, if a person presses *Button 1* or *Button 3* or stand near *photoelectric sensor 1*, Door 1 will open again. After Door 1 is closed completely, Door 2 will open automatically. Each door has two travel switches to detect if the door is opened or closed completely. The most important property which doubles door control system should hold is the two doors cannot be opened at the same time.

Fig. 5 shows the structure of the PLC control system. It is composed of five parts, they are PLC controller, person, sensors, Door 1 and Door 2. Person can press *Button* 1 or *Button* 3 outside Door 1 or Door 2. PLC catches the corresponding signal *Button*1 or *Button*3 in the input scan cycle. After some logic computation, open Door 1 command *OpenDoor*1 or open Door 2 command *OpenDoor*2 is sent by PLC to doors. When the door is opened, people can enter. The opening will last 3 s and then closed automatically. The photoelectric sensors can detect whether a person is near the door. If people have gone away from the door and the Boolean value of *sensor*1 and *sensor*1 are still true. The other door will open automatically. The opening also keeps 3 s and then closed. While the door is opening, people are not allowed to press any buttons. It is the same case when first enters Door 2 and Door 1.

3.2. Formal modeling and validation

The controller and its environment are modeled by timed automata. In timed automata networks, twelve signals are defined. They are *Button*1, *Button*2, *Button*3, *Button*4, *Door*1*Closed*, *Door*1*Opened*, *Door*2*Closed* *Door*2*Opened*, *CloseDoor*1, *OpenDoor*1, *OpenDoor*2 and *ColseDoor*2. One signal is shared by two parts for synchronization.

Fig. 6 shows the model of doors. *c*1 presents the status of doors, *c*1= =1 means the door is closed. At the beginning, The door is

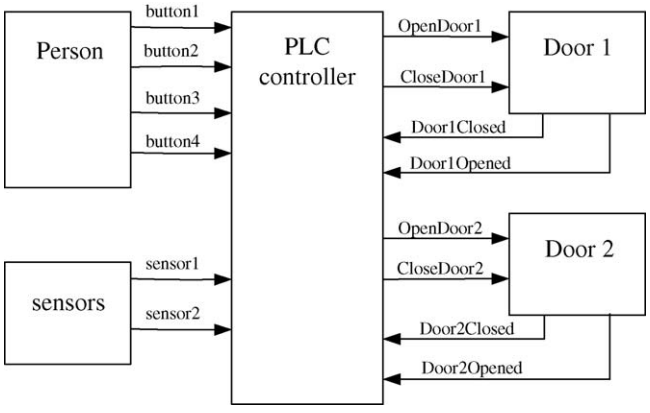

Fig. 5. The structure of double doors controller.

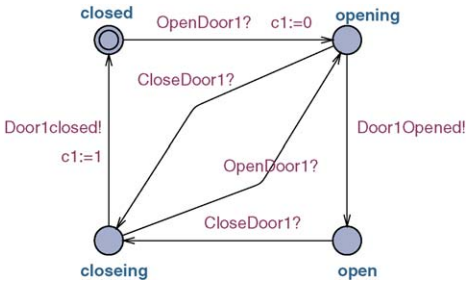

Fig. 6. The model of doors.

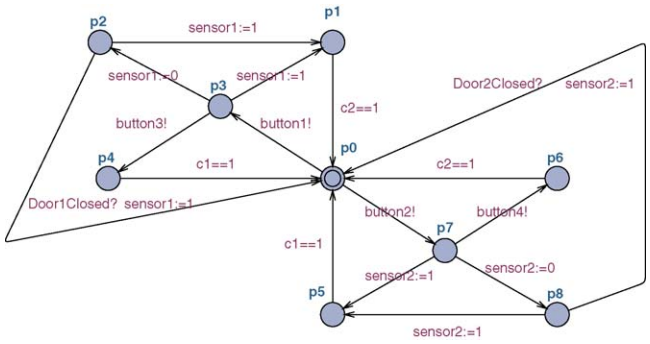

Fig. 7. The behavior model of person.

closed. As soon as it receives a *OpenDoor*1 command, it begin to open. When open completely, the travel switch knows it and send *Door*1*Opened* to PLC. The model moves to *open* state. If it receives *CloseDoor*1 from PLC, the door will close. While travel switch catches the door finishes closing, it sends *Door*1*Closed* and set *c*1 to 1. It is the same for Door 2.

Fig. 7 is the automaton which simulates people's behaviors. It should contain all possible situations. The sensor's model is too simple we compact it with person's model. People can go inside or come outside of the room. The left part is first entering Door 1 and then Door 2. The right part is in the opposite sequence. A person pushed *button*1 and enters the first door. The automaton moves to location *p*3. Trace $p0\ p0 \overset{button1!}{\rightarrow} p3 \overset{button3!}{\rightarrow} p4 \overset{c1==1}{\rightarrow} P0$ means after the person enter the first gate, he does not go through the second door but push *button*3 and go out. *sensor*1 is the value of the Photoelectric Sensor 1. Initially, *senser*1 is 1. When a person stands near the sensor and blocks light, *b*1 turns to 0. So location *p*2 means person enter Door 1 and stands between photoelectric sensors. *c*1 and *c*2 are variables defined in door automata. *c*1= =1 means Door 1 is closed.

The automaton for PLC controller is shown in Fig. 8. The initial location is *M*0. If *button*1 or *button*3 is pressed, it transmits to *M*01. *MuEx*1 and *MuEx*2 are mutual exclusion variables. Then the controller sends open door command *OpenDoor*1 to Door 1, at the same time, *MuEx*2 is evaluated to 0. This means Door 2 can not be opened. As soon as receiving open complete signal *Door*1*Opened*, controller start a timer and go to location *M*2. *t* is a local clock variable. Location *M*2 has a constraint $t <= 3$. As time pass by if $t = = 3$ and *senser*= =1 it transmits to location *M*3 and issues *CloseDoor*1 command. The following trace shows the process of open Door 1.

$$M0 \underset{MuEx1:=1}{\overset{button1?}{\longrightarrow}} M01 \underset{MuEx2:=0}{\overset{OpenDoor1!}{\longrightarrow}} M1 \underset{t:=0}{\overset{Door1Opened!}{\longrightarrow}} (M2, t=0) \overset{\forall c \in \mathbb{R}^+}{\longrightarrow} (M2, t=3) \underset{t==3\ and\ sensor1==1}{\overset{CloseDoor1!}{\longrightarrow}} M3$$

While at location *M*3, there are many choices. Because a person can go ahead through Door 2 or go back through Door 1. Press *button*1 or *button*3 or block photoelectric sensor *sensor*1 will cause Door 1 open again. When controller receives *Door*1*Closed*, it will open Door 2 or go back to initial location depending on the value of *MuEx*1. *MuEx*1=1 indicates Door 1 has finished opening and Door2 will open automatically. Trace

$$M3 \underset{MuEx1==1}{\overset{Door1Closed!}{\longrightarrow}} M02 \underset{MuEx1:=1}{\overset{OpenDoor2!}{\longrightarrow}} M4 \underset{x:=0}{\overset{Door2Opened?}{\longrightarrow}} (M5, x=0) \overset{\forall c \in \mathbb{R}+}{\longrightarrow} (M5, x=3) \underset{x==3\ and\ sensor2==1}{\overset{CloseDoor2!}{\longrightarrow}} M6$$

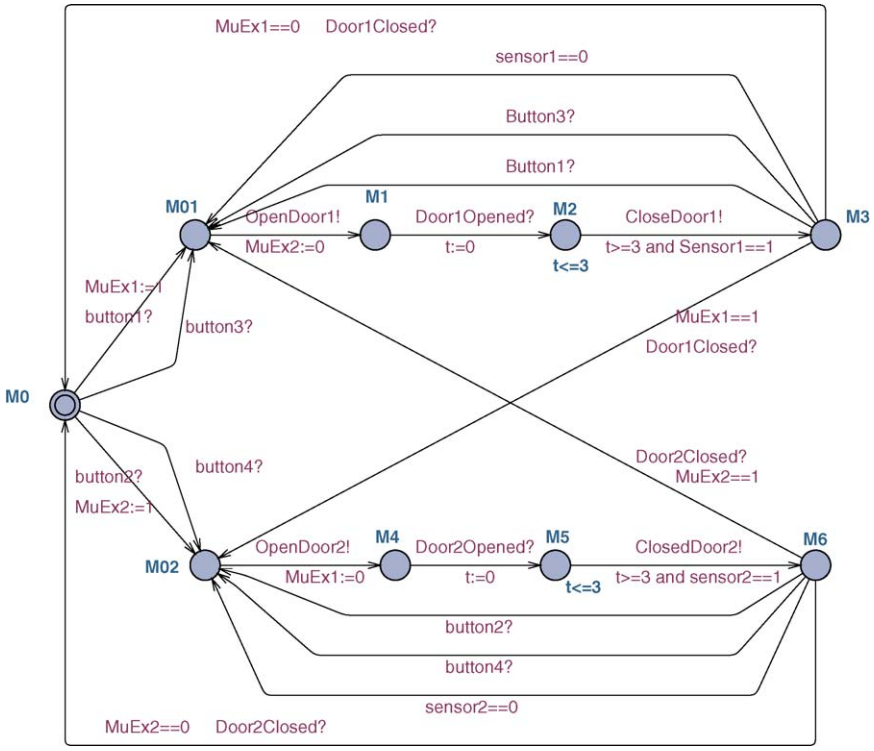

Fig. 8. The model of double door controller.

is the control process of opening Door 2. Because *MuEx*2 has been set to 0 in transition $M01 \xrightarrow[MuEx2:=0]{OpenDoor1!} M1$, the automaton goes back to initial location *M*0. This is an entire process for open Door 1 first and then Door 2. Another trace

$$M0 \xrightarrow[MuEx2:=1]{button2?} M02 \xrightarrow[MuEx1:=0]{OpenDoor2!} M4 \xrightarrow[x:=0]{Door2Opened?} (M5, x=0) \xrightarrow{\forall c \in \mathbb{R}^+} (M5, x=3) \xrightarrow[x==3 \text{ and } sensor2==1]{CloseDoor2!} M6 \xrightarrow[MuEx2==1]{Door2Closed?} M01 \xrightarrow[MuEx2:=0]{OpenDoor1!} M1 \xrightarrow[t:=0]{Door1Opened!} (M2, t=0) \xrightarrow{\forall c \in \mathbb{R}^+} (M2, t=3) \xrightarrow[t==3 \text{ and } sensor1==1]{CloseDoor1!} M3 \xrightarrow[MuEx1==0]{Door1Closed?} M0$$

deals with coming out of the room from Door 2 to Door 1.

The following properties are specified based on the control system requirements. They are verified by Uppaal, run on a computer with PM 750 1.86 GHz CPU with 1.2 GB memory.

P1: The system should be deadlock free.

$A[]$ *not deadlock*

P2: Door 1 and Door 2 can not be opened at the same time

$A[](c1 == 1 \text{ or } c2 == 1)$

All the above properties are satisfied by the control model.

3.3. Code synthesis

We assume *TA* includes $L = \{l_0, l_1, \ldots, l_n\}$, $E = \{e_1, e_2, \ldots, e_m\}$. The Translation method is as follows:

1. Every PLC program has a symbol table which is about the relation of symbols in design model and absolute addresses of PLC hardware. At first we should allocate data address in PLC memory to variables in timed automata. The input signals of PLC correspond to input points *I*0.0, *I*0.1. In our example, *button*1, *button*2, *button*3, *button*4, *Door*1*Closed*, *Door*1*Opened*, *Door*2*Closed*, *Door*1*Opened*, *sensor*1 and *sensor*2 are separately assigned to I0.0, I0.1. The output signals of PLC are mapped to output points *Q*0.1, *Q*0.2. Bit memory areas (M memory) are always used as control relays to store the intermediate status of operations. They can be accessed in bits, bytes, words or double words. Here locations are assigned to M 0.0, M 0.1. of PLC M memory. When automaton arrives at a location, this location variable is set to 1. When automaton leaves this location, variable is reset to 0. We should distinguish the current location and the next location. All newly assigned locations are next locations. We use l_i to present the next value of location l_i. For the concise of program, we use S–R flip flops to describe locations. Another memory areas *V* stores intermediate results of operations. So integer variables can be assigned to *V* memory. The symbol table of double doors example is displayed in Table 1.
2. Edge $l_i \xrightarrow{g,a,r} l_j$ with source location l_i, target location l_j, when guard $g \in \psi$ is satisfied, the transition happens and a subset of variable in *V* are updated by formula $r \in U$.
   - Guard *g* and received signal *a*? are preconditions for the target location variable to be 1. Guard *g* for integer variable is always an inequality. In PLC the corresponding instructions are $>=I$, $<=I$, $>I$, $<I$, $<>I$ or $=I$. Usually, guard *g* is about Boolean variable, ==0 can be implemented by a normally close contact and ==1 match along with normally open contact.

Table 1
The symbol table of double doors examples.

Symbol	Meaning	Address
*button*1	press button 1 outside Door 1	I0.0
*button*2	press button 2 outside Door 2	I0.1
*button*3	press button 3 inside Door 1	I0.2
*button*4	press button 4 inside Door 2	I0.3
*Door*1*Closed*	Door 1 is closed	I0.4
*Door*1*Opened*	Door 1 is Opened	I0.5
*sensor*1	photoelectric sensor 1	I0.6
*sensor*2	photoelectric sensor 2	I0.7
*Door*2*Closed*	Door 2 is closed	I1.0
*Door*2*Opened*	Door 2 is opened	I1.1
*CloseDoor*1	close Door 1 command	Q0.0
*OpenDoor*1	open Door 1 command	Q0.1
*CloseDoor*2	close Door 2 command	Q0.2
*OpenDoor*2	open Door 2 command	Q0.3
*M*0	location for both doors are closed	M0.0
*M*01	location for Door 1 is ready to open	M0.1
*M*1	location for Door 1 is opening	M0.2
*M*2	location for Door 1 has opened	M0.3
*M*3	location for Door 1 is closing	M0.4
*M*02	location for Door 2 is ready to open	M0.5
*M*4	location for Door 2 is opening	M0.6
*M*5	location for Door 2 has opened	M0.7
*M*6	location for Door 2 is closing	M2.0
*M*0′	next value of location M0	M1.0
*M*01′	next value of location M01	M1.1
*M*1′	next value of location M1	M1.2
*M*2′	next value of location M2	M1.3
*M*3′	next value of location M3	M1.4
*M*02′	next value of location M02	M1.5
*M*4′	next value of location M4	M1.6
*M*5′	next value of location M5	M1.7
*M*6′	next value of location M6	M2.1
*MuEx*1	mutual exclusion variable	M2.2
*MuEx*2	mutual exclusion variable	M2.3

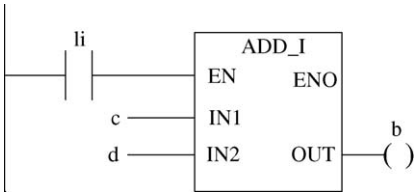

Fig. 9. The LD code for $b := c + d$.

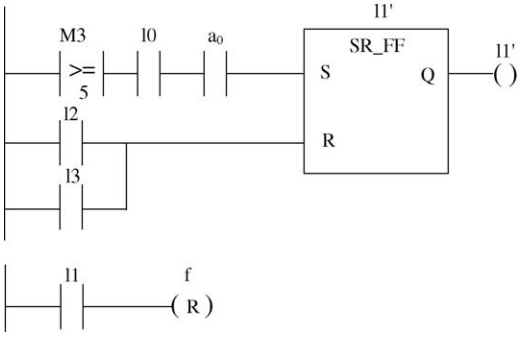

Fig. 10. The LD code for location l_1.

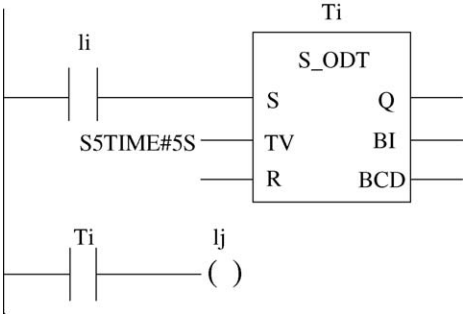

Fig. 11. The LD code for timer T_i.

- Synchronization signals are projected to Boolean variables. So received signal a? is translated to an open contact as the precondition of variable l_j. The sending signal b! means Boolean variable b is set to 1 while this transition is taken. So it is translated to a set coil with address assigned to b. An normally close contact l_i, the guard g and an open contact l_j are the input of coil b.
- Reset formula r of this edge is rewritten as a separate instruction with a normally open contact of source location l_i as the precondition. For Boolean variable, we use R or S instructions and *MOVE* for integer variables. Reset formulas can also have the form $b := c * d$ where * can be +, −, ×, ÷. These formulas corresponds to *ADD_I*, *SUB_I*, *MUL_I* and *DIV_I*. The instruction parameter *IN*1, *IN*2 and *OUT* are assigned to c, d

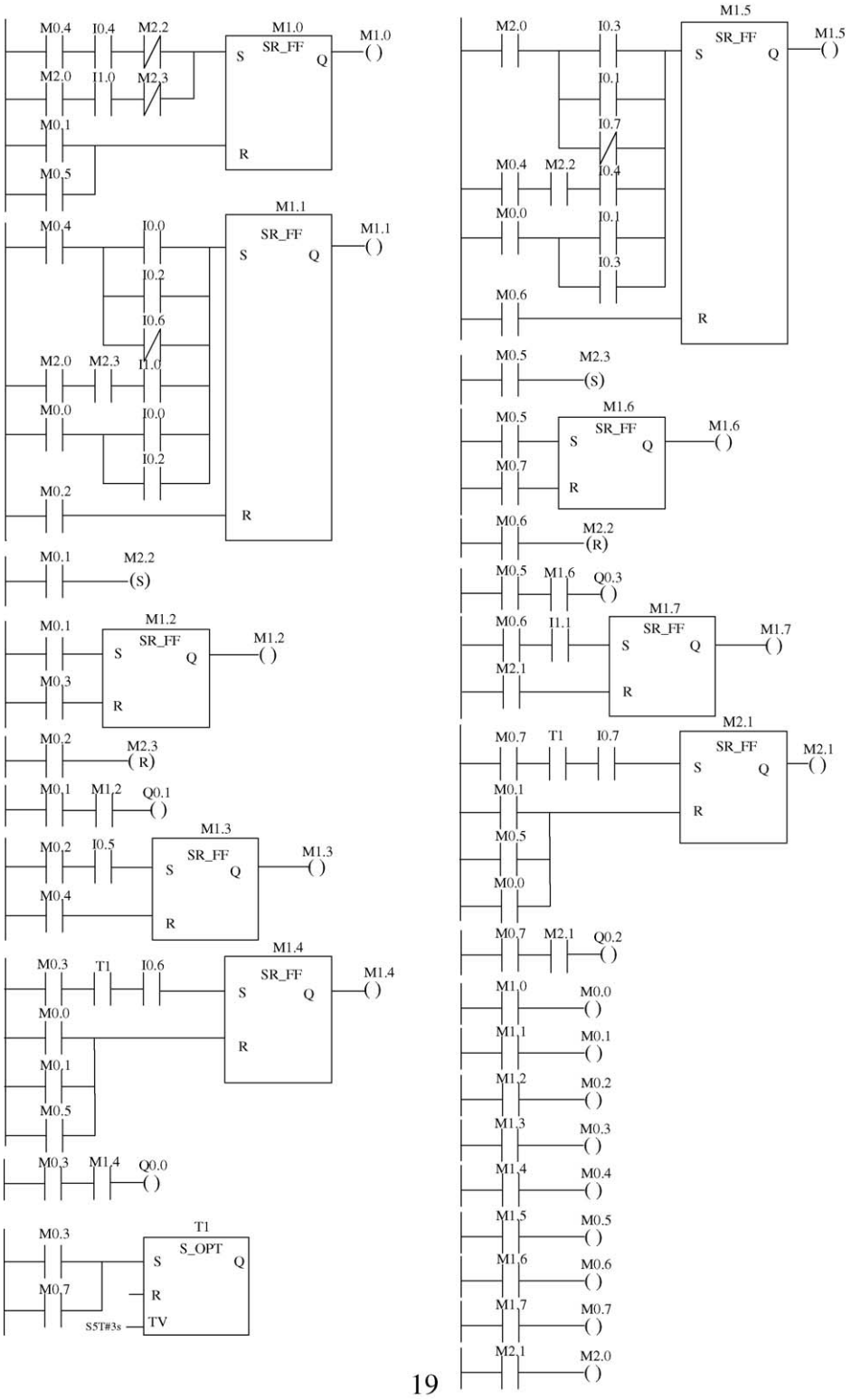

Fig. 12. The LD code for double door controller.

and b separately. These instructions are all activated by a normally open contact l_i which is connected to *EN* point. We can see in Fig. 9.

3. After this transition the automaton leaves location l_i and arrives at l_j. Correspondingly the next value l_i of location l_i is set to 0 and l_j to 1. So the disjunction of all target locations are preconditions for resetting *SR* flip-flop l_j. In other words, the incoming edges of a location decide the set precondition and outgoing edges decide the reset precondition.

   For example, location l_1 has related transitions $l_0 \xrightarrow{c>=5, a_0?, f:=0} l_1$, $l_1 \xrightarrow{g_1, a_1!, r_1} l_2$ and $l_1 \xrightarrow{g_2, a_2!, r_2} l_3$. Instructions about l_1 can be translated as LD program in Fig. 10.
4. Timer is a special part of PLC programs. PLC has three kinds of timers, they are On-Delay Timer, Retentive On-Delay Timer and Off-Delay Timer. Time memory area reserves one 16-bit word for each timer address. The maximum number of PLC Timer is 256. This can meet the requirements of applications. There are three different solutions of timers: 10 ms, 100 ms and 1 s. The serial number of Timer determines the solutions. Bit 12 and 13 of timer words decide the classification and resolution. On-Delay Timer is power enough to model clocks in TA. Bit 0-11 hold the time value in BCD format.

   One clock variable corresponds to a separate timer block in PLC. The outgoing transition of time location always has a guard about clock variable c. The normal form is $c >= n$ where $n \in N$. In order to enable the transition we add function $c <= n$ as the constraint function for time location. We call locations with clock constraints *time location*. In Uppaal, clock variables are declared separately. When exploring the timed automata of PLC, and meeting a clock variable c_i in location l_i, we start a timer T_i. The enable condition (S input) is location variable l_i where the elapses of time transitions happens. The preset time value (TV) of T_i is related to the guard function $c_i >= n$ of the outgoing edges to location l_j. TV is set to n. The timer is reset if the R input changes from 0 to 1. The output Q turns to 1 when timer value reaches TV. Timer T_i is the precondition for the target location l_j. The LD code is in Fig. 11.

At the end of this computation cycle, the next value of locations are assigned to current variables for next cycle computation.

We design an algorithm to implement all the above rules. Timer is the specifical part of the program. Before applying this algorithm, we should select proper Timer for time location considering time resolution. According to the clock guard (T_k *value*) of transitions, we fix preset value of Timer (T_k *TV*) first. The *S* port of Timer is the location variable. Then all locations are assigned to M memory addresses. So location symbols can appear in instructions. The set and reset operations for location variables is implemented by S–R flip flop. *Reset*(v) is the R input and *Set*(v) is the S input of SR instruction with address v. *forward*(v) records all target locations of location v. *mark*(l) = 1 means location l has been visited. *father*(l) records the source locations of l. *ReSig*() and *SeSig*() stand for received and sent signals of transitions. *Enable*() is the *EN* input port of compare and math instructions.

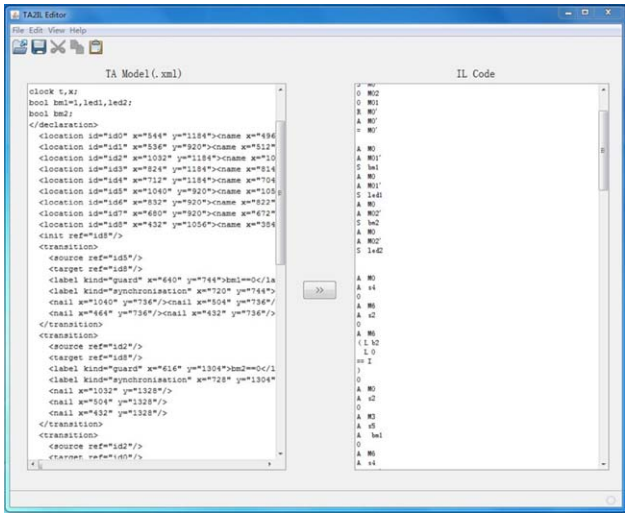

Fig. 13. Tool TA2IL.

Algorithm 1 Automatic code synthesis

1: **for all** location $l \in L$ **do**
2: $mark(l) \leftarrow 0$
3: $father(l) \leftarrow 0$
4: $forward(l) \leftarrow \Phi$
5: **end for**
6: **for** The initial location l_0 **do**
7: $mark(l_0) \leftarrow 1$
8: $v \leftarrow l_0$
9: **end for**
10: **if** All edge came from v is visited **then**
11: $Reset(v') \leftarrow \vee(forward(v))$
12: goto 28
13: **else**
14: choose one unvisited edge $e_j(v \rightarrow w)$
15: $forward(v) \leftarrow \{w\} \cup forward(v)$
16: $Set(v') \leftarrow \vee(ReSig(e_j) \wedge w \wedge guard(e_j))$
17: $Set(SeSig(e_j)) \leftarrow v$
18: $Enable(UpDate(e_j)) \leftarrow v$
19: **if** v is time location **then**
20: $T_k.S \leftarrow v$
21: $T_k.TV \leftarrow T_kValue$
22: **end if**
23: **if** $mark(w) = 0$ **then**
24: $mark(w) \leftarrow 1, v \leftarrow w, father(w) \leftarrow v$
25: **end if**
26: **end if**
27: goto 10
28: **if** $father(v) \neq 0$ **then**
29: $v \leftarrow father(v)$ and goto 10
30: **else**
31: **return**
32: **end if**
33: **for all** location $l \in L$ **do**
34: $l \leftarrow l'$
35: **end for**

The time complexity of this algorithm is $O(\max\{|L|, |2E|\})$ where $|L|$ is the number of locations and $|E|$ is the number of edges.

Fig. 12 shows the code generated for the double door controller. We simulated these codes in PLCSIM and they correctly satisfy the

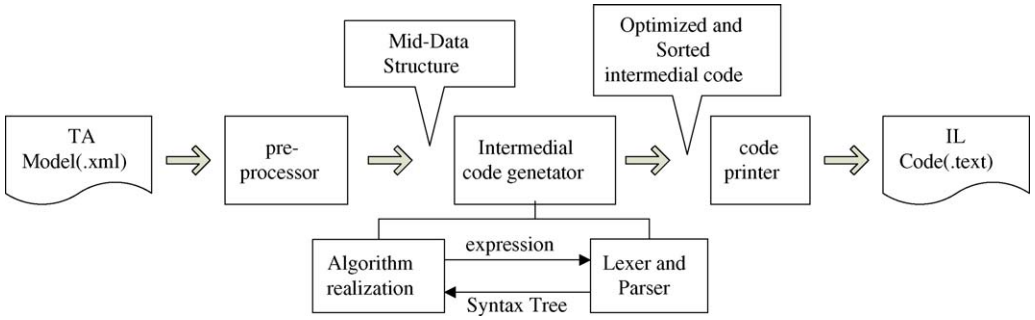

Fig. 14. The structure of tool TA2IL.

specifications. We will explain how to get these codes though the algorithm.

$L = \{M0, M01, M1, M2, M3, M02, M4, M5, M6, \}$

Let us look closer at transitions about location $M0$. $M01$ and $M02$ are the target locations of M0. We have $Forward\ (M0) = \{M01, M02\}$. The conjunction of $M01$ and $M02$ is the R input of S–R flip flop for $M0$. $M0$ has two incoming edges:

$$Income(M0) = \{M3 \xrightarrow[MuEx1==0]{DoorClosed?} M0, \quad M6 \xrightarrow[MuEx2==0]{Door2Closed?} M0\}$$

$MuEx1$ and $MuEx2$ are Boolean variables, so $MuEx1 == 0$ and $MuEx2 == 0$ are mapped to closed contacts. One condition for set flip flop $M0$ is the conjunction of received signal $Door1Closed$ and guard $\overline{MuEx1}$ together with $M3$. The other one is the conjunction of received signal $Door2Closed$, guard $\overline{MuEx2}$ and $M6$. So we have:

$Reset(M0') \leftarrow M01 \vee M02$
$Set(M0') \leftarrow (M3 \wedge Door1Closed \wedge \overline{MuEx1}) \vee (M6 \wedge Door2Closed \wedge \overline{MuEx2})$

Then the algorithm selects $M01$ as the next location. $forward(M01) = \{M1\}$, So we have:

$Reset(M01') \leftarrow M1$

$M01$ has six incoming edges:

$$Income(M01) = \{M0 \xrightarrow[MuEx1:=1]{Button1?} M01,\ M0 \xrightarrow{button3?} M01,\ M3 \xrightarrow{button3?} M01,\ M3 \xrightarrow{button1?} M01,\ M3 \xrightarrow{sensor1==0} M01,\ M6 \xrightarrow[MuEx2==1]{Door2Closed?} M01\}$$

So we have:

$$Set(M01') = (M3 \wedge button1) \vee (M3 \wedge \overline{button3}) \vee (M3 \wedge \overline{sensor1}) \vee (M6 \wedge MuEx2 \wedge Door2Closed) \vee (M0 \wedge button1) \vee (M0 \wedge button3)$$

None of the incoming edges have sent signal and update formulas.

For $M1$, $forward(M1)$={$M2$}, and $Income(M1) = \{M01 \xrightarrow[MuEx2:=0]{OpenDoor1!} M1\}$ has a update formula $MuEx2$:=0 and sent message $OpenDoor1$. We get the instructs:

$Reset(M1') \leftarrow M2$
$Set(M1') \leftarrow M01$
$Reset(MuEx2) \leftarrow M1$
$Set(OpenDoor1) \leftarrow M1$

Table 2
The result of examples.

Example	Property satisfied	Number of instructions
Motor controller	3	25
Quiz machine	4	26
Steeve controller	6	61

$M2$ and $M5$ are timing locations. As soon as automaton arrives at these locations, timer is enabled. As the boundary time for transitions are both 3s. We compact two timers as one. The value of T_1 TV is $S5T\#3s$ and the value of T_1S is $M2 \times M5$. The pseudo code is:

$Set(T1) \leftarrow M2 \vee M5$
$T1.TV \leftarrow S5T3S$

3.4. Implementation

We have implemented a prototype tool TA2IL of the above algorithm showed in Fig. 13. Timed automata designed and verified in model checker Uppaal are stored as XML files. In order to connect these tools seamless, the XML descriptions of timed automata are the input format of TA2IL.

Fig. 14 shows the main components of this tool. Pre-processor parses the XML files of UPPAAL, and saves location and transition information by pre-defined data structure. Code generator explores controller automaton and sorts the locations with depth first order. It combines the properties for the same location and throw off redundant data. It analyzes expressions of transitions with the help of ANTLR[15], and get a syntax tree according to our algorithm. Code printer first sorts all the bit memory addresses, then optimizes and merges all the Set and Reset condition for these variables. As LD is a graphic language and IL is a text language. IL is more fundamental. All LD program can change to IL. The output of this tool is IL description for LD program.

We have designed and implemented other examples using this method. In the premise of a correct design model, the generated programs satisfy the system requirements. The experiment results are displayed in Table 2.

4. Conclusions

In this paper, we demonstrated a formal developing process for reliable PLC software. Timed automata are used as the formal models in design and analysis process. With the help of model checker, the design models satisfy user requirements. We proposed a PLC code synthesis algorithm based on this formal model. This algorithm has been implemented with a prototype tool TA2IL. Our method can condense the developing process. The case study showed the effectiveness of the method.

Acknowledgements

This work was partly supported by the Chinese National 973 Plan under grant No. 2010CB328003, the NSF of China under project No.90718039.

References

[1] http://www.uppaal.com (accessed January 2009).
[2] G. Behrmann, A. David, K.G. Larsen, A tutorial on Uppaal, in: Proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, LNCS, vol. 3185, 2004.
[3] G. Behrmann, J. Bengtsson, A. David, K.G. Larsen, P. Pettersson, W. Yi, Uppaal implementation secrets, in: Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems, 2002.
[4] R. Wang, M. Gu, X Song, Modeling and verification of program logic controllers with timed automata, IET Software 1 (4) (August 2007) 127–131.
[5] K. Sacha, Automatic code generation for PLC controllers, in: R. Winther, B.A. Gran, G. Dahll (Eds.), SAFECOMP 2005. LNCS, vol. 3688, Springer, Heidelberg, 2005, C303–C316.
[6] K. Sacha, Translatable Finite State Time Machine. SDL 2007: Design for Dependable Systems, LNCS, vol. 4745, LNCS, Berlin/Heidelberg, 2007, pp. 117–132.
[7] K. Sacha, Verification and implementation of dependable controllers, in: Proceedings of the Third International Conference on Dependability of Computer System DepCoS-RELCOMEX, 2008, pp. 143–151.
[8] S. Klein, G. Frey, M. Minas, PLC Programming with Signal Interpreted Petri Nets, in: Proceedings of the International Conference on Application and Theory of Petri Nets (ICATPN 2003), Eindhoven (The Netherlands), LNCS, vol. 2679, Springer, June 2003, pp. 440–449.
[9] D. Pollmacher, W. Zimmermann, H.M. Hanisch, Translation validation for model-based code-generators for PLCs, Emerging Technologies and Factory Automation (2005) 113–120.
[10] International Electrotechnical Commisson, Techincal Committee No. 65. Programmable Controller-Programming Languages, IEC61131-3, second edition (1998), comminttee draft.
[11] Ladder Logic for S7-300 and S7-400 programming. Reference Manual, 2002.
[12] R. Alur, D.L. Dill, A theory of timed automata, Theoretical in Computer Science 126 (2) (1994) 183–235.
[13] R. Alur, Timed automata, in: Proceedings of Computer Aided Verification, Trento, IT, LNCS, vol. 1633, Springer-Verlag, July 1999, pp. 8–22.
[14] J. Bengtsson, W. Yi, Timed automata: semantics algorithms and tools, in: Lecture Notes on Concurrency and Petri Nets. LNCS, vol. 3098, Springer-Verlag, 2004.
[15] T. Parr, The Definitive ANTLR Reference, The Pragmatic Bookshelf, 2007.

Rui Wang is a Ph.D candidate in Department of Computer Science and Technology at Tsinghua University. She received her bachelor's degree in Xi'an Jiaotong University, China, 2004. Her research interests are embedded system modeling and formal methods

Xiaoyu Song received the Ph.D. degree from the University of Pisa, Italy, 1991. From 1992 to 1999, he was on the faculty at the University of Montreal, Canada. In 1998, he worked as a Senior Technical Staff in Cadence, San Jose. In 1999, he joined the faculty at Portland State University. He is currently a Professor in the Department of Electrical & Computer Engineering at Portland State University, Portland, Oregon. His current research interests include formal methods, design automation, embedded system design, and emerging technologies. He has been awarded as the Intel Faculty Fellow during 2000–2005. He served as an associate editor of IEEE Transactions on Circuits and Systems and IEEE Transactions on VLSI Systems.

Jianzhong Zhu is a Master student in the School of Software at Tsinghua University, China. He completed his Bachelor Degree from Jiangsu University. His area of research is related to software engineering and automation.

Ming Gu, Researcher, Vice-director, School of Software, Tsinghua University, Vice-director of Ministry of Education Key Laboratory of Information Security. Research areas include software formal methods, software trustworthy, middleware technology.