Formalization and Verification of PLC Timers in Coq

Hai Wan, Gang Chen, Xiaoyu Song, Ming Gu

Formalization and Verification of PLC Timers in Coq

Hai Wan
Key Lab for ISS of MOE, School of Software, Tsinghua University, Beijing, China
Email: wanh03@mails.tsinghua.edu.cn

Gang Chen
Lingcore Laboratory Portland, USA
Email: gangchensh@gmail.com

Xiaoyu Song
ECE Dept, Portland State University Portland, USA
Email: songpisa@yahoo.com

Ming Gu
Key Lab for ISS of MOE, School of Software, Tsinghua University Beijing, China
Email: guming@tsinghua.edu.cn

***Abstract*—Programmable logic controllers (PLCs) are widely used in embedded systems. A timer plays a pivotal role in PLC real-time applications. The paper presents a formalization of TON-timers of PLC programs in the theorem proving system Coq. The behavior of a timer is characterized by a set of axioms at an abstract level. PLC programs with timers are modeled in Coq. As a case study, the quiz machine problem with timer is investigated. Relevant timing properties of practical interests are proposed and proven in Coq. This work unveils the hardness of timer modeling in embedded systems. It is an attempt of formally proving the correctness of PLC programs with timer control.**

***Keywords*-PLC; TON-Timer; Modeling; Coq**

I. INTRODUCTION

Programmable logic controllers (PLCs) are widely used for safety critical applications in various industrial fields. The correctness and reliability of PLC programs are of great importance. The use of timers is one of the distinguished features of PLCs programs, since the notion of time is mainly introduced by timers in PLC systems. Hence, the main focus of this paper is the modeling and verification of PLC programs with timers.

Based on the operational semantics, in [1] PLC programs are translated into timed automata. They proposed two ways to model timers: one is to treat timers as symbolic function block calls and the other is to model timers as separate timed automata. Model checker UPPAAL is employed to verify the model. In [3], timed automata are used to model Sequential Function Chart programs. In their work, the use of timers is restricted – each step is associated with a timer and the timer is taken for guarding the transitions – in other words, timers are not used in a restricted form. A special automaton that orients real-time systems – PLC-automaton is developed to model the specifications of real-time applications. Structured Text programs can be automatically generated from PLC-automata [4]. They did not model PLC programs with timers, but use timers to implement PLC-automata. In [5] condition/event systems are adopted to model PLC programs. There is an assumption that timers can be started only at the beginning of the calculation phase. In [6], the Ladder Diagram programs are investigated and time is treated implicitly. Model checker SMV was used to verify the model. The methods mentioned above are all related to the model checking technologies.

Besides model checking, theorem proving is alsoemployed in verification of PLC Programs. In [7] the theorem system Isabelle/HOL was used to model and verify PLC programs. Modular verification method is adopted in the paper. They had a simple model of time, because they assumed that the current value increases monotonously and there is no reset action during this process. No explicit model of a timer was given. In [8], the synchronous language SIGNAL is used to model Structured Text and Function Block Diagram programs. They did not treat timer instructions.

This paper is an attempt of formally proving the correctness of PLC programs with timer control. The Coq [2] theorem prover is chosen as our formal verification tool. It allows problem specification, program formalization and property proving in a single working environment. A timed quiz machine problem is employed as the case study example.

Informally, the main properties under investigation are statements of the forms that, if a proper sequence of stimuli are received by the PLC program, then some expected outcomes will be observed. The problem becomes involved as the timer input depends on its output in previous cycle. By the nature of the TON timer, its input signals have to be kept stable during the timing period. That is, their values have to be constant between the start of timing process and the timeout point. However, the cyclic program structure does not make this property obvious. To make the proving process manageable without loss of generality, we will introduce three abstract axioms at an appropriate level to characterize

Hai Wan and Ming Gu are with the Tsinghua National Laboratory for Information Science and Technology (TNList), and School of Software, Tsinghua University, and the Key Laboratory for Information System Security, Ministry of Education, Beijing, China.

This work was supported in part by the Chinese National 973 Plan under grant No. 2004CB719400, the NSF of China under grants No. 60553002, 60635020 and 90718039.

the behavior of TON timer (see section IV for details).

As the first step in this formalization, we assume the existence of a function f which maps the start of each scan cycle to its time point. This function is abstract and it needs only to satisfy a monotonic requirement. Such an assumption allows us to establish the relation between scan cycles (see next section for the descriptions of PLC and the notion of scan cycle) and real time points without explicitly calculating the exact time period of each cycle. It should be noted that the adoption of this function relies on the programming practice that the timer output is sampled only once in the program. A subtle issue in this formal model is the selection of the starting and ending points of each scan cycle. In the PLC convention, a scan cycle starts with the input phase, followed by the execution phase and the output phase. As a result, the definition of the function f encounters many choices. It can map the start of the input phase to time, or map the start of the execution phase or the start of the output phase to time. Such a selection would give different interpretations to timer axioms and will influence the forms of system properties. A further discussion on this issue can be found in section IV-B and IV-E.

With the preparation given above, we can give an informal description on the axiomatic assumptions of TON timer:

- if the main input to the timer turns off, so does the timeout signal;
- if the preset timeout period is passed and that the main timer input has been kept ON during this period, then the timeout signal will turn on (or keep on) at next cycle;
- if the timeout signal will be turned on at next cycle, then the main input signal must have been kept on for a period larger than the preset timing period.

The precise description of these hypothesis is presented in section IV-E2. These assumptions characterize the global behavior of TON timer. They appear both reasonable and sufficient for the verification of most timer-related problems.

From this study, it made clear that formal PLC verification can benefit from Coq in many aspects. First, Coq gives us enough expressive power to model PLC programs with timers at any desired abstract level. A formal timer model, expressed in a set of Coq axioms or hypothesis, can abstract away the differences among timers with different resolutions. The abstraction facilitates the modeling and verification process. Second, Coq allows us to prove parameterized properties, which is a useful feature not directly supported by model checking. Third, it is easy to model objects by parameterized Coq modules so that specifications and proofs can be reused and scalable.

The rest of the paper is organized as follows. In the following section, we give a short introduction to PLC and its timers. A quiz machine example is described in Section III. The example is employed throughout the paper. In Section IV, we describe the method of modeling timers in the theorem prover Coq in detail. Section V shows the complete model of the example program and outlines the proof. Finally, Section VI concludes the paper.

II. Programmable Logic Controller and Timers

A PLC system typically consists of a CPU, a memory and input/output points through which the system communicates with its outside environment. The execution of a PLC program is an infinite loop in which each iteration is called scan cycle. Typically, each scan cycle can be divided into three phases:

1) **Input phase** during which PLC system reads the values of sensors and copies them into memory which forms a snapshot of the environment. These values do not change throughout the rest of the cycle. Input phase takes the same time in each scan cycle.
2) **Calculation phase** during which the PLC system executes the instructions and writes back the results into memory. At the beginning of each calculation phase, PLC system does some preparations, such as self-check and timer instructions (for the timers whose base is 10ms). The preparations take the same time.
3) **Output phase** during which PLC maps the results into actuators. Output phase takes the same time in each scan cycle.

There are five standard programming languages for PLC [9] among which the Ladder Diagram (LD) is the most widely used programming language. From now on, we concentrate on the LD language.

As an embedded system, the real-time aspect of PLC system is ensured by the use of timers. [1] There are mainly three kinds of timers in S7-200[10]: TON-timer, TONR-timer, and TOF-timer. In this paper, we focus on the TON-timers. The other two kinds of timers can be treated in a similar manner. A TON-timer has two input ports: IN that indicates whether the timer is enabled and PT that is the preset value of the timer. There are two output ports: one for the current value and the other for the timer bit. The characteristics of a TON-timer are informally described as follows:

1) A TON-timer counts time (i.e. increase its current value) when its IN is ON and it is updated.
2) A TON-timer's timer bit is OFF and its current value is set to zero when its IN is OFF.
3) If a TON-timer's current value is greater than or equal to its PT, its timer bit is ON.
4) A TON-timer continues counting after its PT is reached, and stops counting when the maximum value 32767 is reached.

[1] Since there are many different kinds of PLCs in the industry and the instructions used in these PLCs are different from each other, in order to ease the discussion, from now on, we focus on S7-200 which is a kind of PLC produced by the Siemens company. In some cases, time is ensured by the use of interruptions. We do not consider this case here.

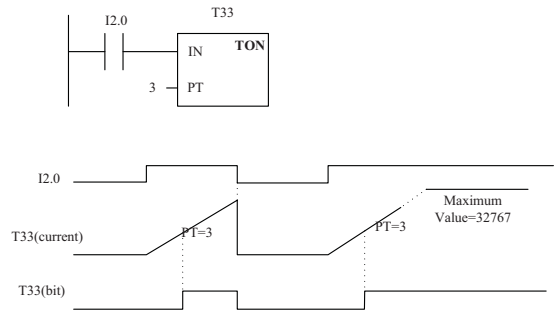

Figure 1. TON timer and its behivour

According to the user manual of S7-200, a TON-timer and its behavior are demonstrated in Fig.1.

A TON-timer has three resolutions: 1ms, 10ms, and 100ms. The behaviors of timers with different resolutions are different. For the timer with time base of 1ms, it updates every 1ms asynchronously to the scan cycle. In other words, the timer bit can be updated multiple times during one scan cycle. For the 10ms timer, it only updates at the beginning of each scan cycle. For the 100ms timer, its current value is updated only when the instruction is executed. The user manual emphasizes that, for the 100ms timer, its instruction should be executed one and only one time during each cycle.

In section IV-E, we propose a set of axioms to model the behavior of a TON-timer. This axiom set appears sufficient for proving many properties of practical interests.

III. An Illustrative Example

In this section, we show a quiz machine problem as an illustrative example to explain some basic notions of LD language, control flow graph of program, and the modeling process of TON-timers.

A quiz machine is an equipment used in a contest which involves a host and several players. The host uses his buttons to start and reset a contest. Every player controls his button which is associated with a light. The button is used for the player to vie to answer and the light is used to indicate that the corresponding player has the chance to answer. After the host starts a contest, the first player who presses his button within the predefined time will turn on an associated light. If more than one players press the buttons at the same time, the machine should inform the host to restart another contest. If during the predefined time there is no one pressing the button, the machine should inform the host that time is out and keep all players' lights off even if some of them press their buttons. The ladder diagram implementation of quiz machine with three players and a predefined time of 3 seconds are shown in Fig.2.

As shown in Fig.2, a LD program consists of a set of rungs. There are 10 rungs in the example program. Each rung can be seen as a connection between logical checkers (contacts or relays) and actuators (coils). There are two kinds of relays: normally open contact (-| |-) and normally closed contact (-|/|-). Each relay or coil is associated with a bit which can be 0 or 1. For example, the first rung contains three relays (which are associated with bits m_1, i_1 and i_0 respectively) and one coil (associated with bit m_1). Normally, each rung has only one coil. If a path can be traced between the left side of the rung and the coil, through ON relays, the rung is true and the bit of output coil is 1. A normally open relay is ON iff its bit is 1. A normally closed relay is ON iff its bit is 0. Intuitively, each rung can be understood as a assignment consists of the bits in the rung. For example, the first rung can be expressed by $m_1 = (i_0 \wedge m_1) \vee \neg i_1$.

In the program, the start and reset buttons are associated with i_0 and i_1 respectively (i.e. if start button is pressed then i_0 is 1). The buttons for players are i_2, i_3 and i_4. o_1, o_2, and o_3 are used to control the corresponding lights l_1, l_2, and l_3, respectively. o_0 denotes whether the time is out. The system's inner states consist of five bits: from m_1 to m_5. m_1 indicates whether the contest begins. TON-timer t_1 counts when m_1 is 1 and is used to record the escaped time after the host presses start button. The timer bit of t_1 is 1 when the escaped time extends the predefined timeout. m_2, m_3 and m_4 denote whether play_1, play_2 or play_3 first presses his button respectively. m_5 represents whether there is a player that presses his button within the predefined timeout.

Three time related program properties should be satisfied. The host presses the reset button and, after a while, he presses the start button. Between the time he presses the start button and the timeout:

- if there is only one who first press his button, the corresponding light will turn on and the other lights stay off;
- if at least two players first press their buttons at the same, their lights will be turned on and the other lights stay off;
- if no one presses the button, the light indicating timeout will turn on and the other lights stay off.

IV. Modeling PLC Programs with Timers

Following the criteria proposed in [11], we show our method of modeling PLC programs with timers in three steps:

1) programming language fragments that are used and assumptions and constraints of the PLC programs; we also propose a preprocess that makes PLC programs satisfy the constraints.
2) how to model the cyclic operation mode;
3) how to model the timer.

We first introduce the control flow graph and its associated table which give us a formal base for the description and verification of constraints.

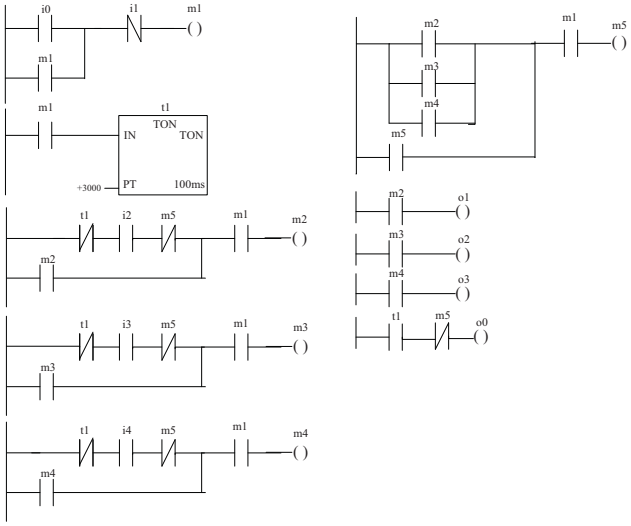

Figure 2. The Ladder Diagram of Quiz Mechine

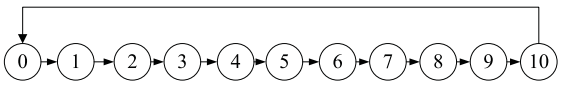

Figure 3. The CFG of the quiz machine program

Table I
refS AND defS FOR THE CFG OF QUIZ MACHINE

Node No.	ref	def	$times$
0	$\{\}$	$\{\}$	{0ms}
1	$\{i_0, i_1, m_1^1\}$	$\{m_1\}$	{3ms}
2	$\{m_1^1\}$	$\{t_1\}$	{6ms}
3	$\{t_1^2, i_2, m_5^6, m_1^1, m_2^3\}$	$\{m_2\}$	{9ms}
4	$\{t_1^2, i_3, m_5^6, m_1^1, m_3^4\}$	$\{m_3\}$	{12ms}
5	$\{t_1^2, i_4, m_5^6, m_1^1, m_4^5\}$	$\{m_4\}$	{15ms}
6	$\{m_1^1, m_2^3, m_3^4, m_4^5, m_5^6\}$	$\{m_5\}$	{18ms}
7	$\{m_2^3\}$	$\{o_1\}$	{21ms}
8	$\{m_3^4\}$	$\{o_2\}$	{24ms}
9	$\{m_4^5\}$	$\{o_3\}$	{27ms}
10	$\{t_1^2, m_5^6\}$	$\{o_0\}$	{30ms}

A. Control Flow Graph of PLC Programs

The structure of a PLC program can be described by a control flow graph (CFG). As an example, the CFG of the quiz program is shown in Fig.3. There is a special node N_0 that represents the preparation phase at the beginning of the calculation phase. Every program's CFG has such a node, since every scan cycle contains a preparation phase. Each of the rest nodes denotes a rung in the program. Since there is no loop and case constructs in the program, the CFG is a simple cycle (note the cyclic behavior of PLC systems). Every node i is associated with three sets: ref_i, def_i, and $times_i$, where ref_i is the set of variables node i refers to, def_i is the set of variables node i defines and $times_i$ is the set of all possible time spans used to reach node i from the beginning of the calculation phase. Intuitively, ref_i is the set of variables used in the right hand side of the assignment related to rung$_i$, while def_i is that appear in the left hand side of the assignment. Since each rung has only one coin, the cardinality of def_i is always one. Tab.I shows the sets for each node. The superscript i of each variable in the ref_i means the variable is defined at node i. For instance, m_1^1 in ref_2 means m_1 used at node 2 is defined at node 1. We assume the executions of each rung cost the same time which is 3ms.[2] CFG and the table form a formal representation of a PLC program's structure.

The timer bits are special variables in PLC. Comparing to other variables, their values can change multiple times without explicit assignments. If 1ms TON-timer is chosen for the program in Fig.2, we need to add t_1 to every def_i – since it can update asynchronously to the scan cycle – and make some modifications to t_1^* in each ref_i. The result table is shown in Tab.II.

[2]The execution times for different rungs can be different, but this doesn't effect the modeling and verification process.

B. Assumptions and Constraints

We articulate the assumptions and constraints below and explain how to modify the program that does not meet these properties to make them satisfy these properties:

1) The executions of the same instruction take the same time.
2) There is no loop in one scan cycle, i.e. if we remove N_0, the resulted CFG is an acyclic graph.
3) During one scan cycle, the values of a TON-timer used by several instructions are the same. This can be stated formally as $\forall n_1\ n_2\ n_3\ n_4 : Nodes, t : Timers : t^{n_3} \in ref_{n_1} \vee t^{n4} \in ref_{n_2} \rightarrow n_3 = n_4$. If the program does not satisfy the constraint, this could cause unfair treatment to players in the quiz competition. Given a program does not satisfy the constraint, the following modifications should be made to the program:
   - **1ms TON-timer.** For each 1ms TON-timer t, a new variable m is introduced and a new rung

Table II
refS AND defS FOR THE CFG OF QUIZ MACHINE WITH 1MS TON-TIMER

Node No.	ref	def	$times$
0	$\{\}$	$\{t_1\}$	{0ms}
1	$\{i_0, i_1, m_1^1\}$	$\{m_1, t_1\}$	{3ms}
2	$\{m_1^1\}$	$\{t_1\}$	{6ms}
3	$\{t_1^2, i_2, m_5^6, m_1^1, m_2^3\}$	$\{m_2, t_1\}$	{9ms}
4	$\{t_1^3, i_3, m_5^6, m_1^1, m_3^4\}$	$\{m_3, t_1\}$	{12ms}
5	$\{t_1^4, i_4, m_5^6, m_1^1, m_4^5\}$	$\{m_4, t_1\}$	{15ms}
6	$\{m_1^1, m_2^3, m_3^4, m_4^5, m_5^6\}$	$\{m_5, t_1\}$	{18ms}
7	$\{m_2^3\}$	$\{o_1, t_1\}$	{21ms}
8	$\{m_3^4\}$	$\{o_2, t_1\}$	{24ms}
9	$\{m_4^5\}$	$\{o_3, t_1\}$	{27ms}
10	$\{t_1^9, m_5^6\}$	$\{o_0, t_1\}$	{30ms}

Table III
refs AND defs FOR THE CFG OF MODIFIED QUIZ MACHINE WITH 1MS TON-TIMER

Node No.	ref	def	$times$
0	$\{\}$	$\{t_1\}$	{0ms}
1	$\{i_0, i_1, m_1^1\}$	$\{m_1, t_1\}$	{3ms}
2	$\{m_1^1\}$	$\{t_1\}$	{6ms}
3	$\{t_1^2\}$	$\{m_6\}$	{9ms}
4	$\{m_6^3, i_2, m_5^6, m_1^1, m_2^3\}$	$\{m_2, t_1\}$	{12ms}
5	$\{m_6^3, i_3, m_5^6, m_1^1, m_3^4\}$	$\{m_3, t_1\}$	{15ms}
6	$\{m_6^3, i_4, m_5^6, m_1^1, m_4^5\}$	$\{m_4, t_1\}$	{18ms}
7	$\{m_1^1, m_2^3, m_3^4, m_4^5, m_5^6\}$	$\{m_5, t_1\}$	{21ms}
8	$\{m_2^3\}$	$\{o_1, t_1\}$	{24ms}
9	$\{m_3^4\}$	$\{o_2, t_1\}$	{27ms}
10	$\{m_4^5\}$	$\{o_3, t_1\}$	{30ms}
11	$\{m_6^3, m_5^6\}$	$\{o_0, t_1\}$	{33ms}

"$|--|\ t\ |--(m)$" is inserted before the first reference to t. All the references to t are replaced by the references of m. For example, if 1ms TON-timer is used in the program shown in Fig.2, the program does not satisfy this constraint for the values used at node 3 and 4 are different i.e. in Tab.II the superscripts of t_1 at nodes 3 and 4 are different. The table of the program after modification is shown in Tab.III from which it can be verified that the program holds the constraint.

- **10ms and 100ms TON-timers.** No modification is needed.

4) For each node i in CFG that defines a timer, i.e. the timer is in def_i, and referred by other nodes in the same cycle, the cardinality of $times_i$ is 1. In other words, for such node there is one and only one path to reach it. This constraint ensures that the time intervals used to reach the same timer instruction from the beginning of the scan cycle are the same.
5) Each relay can be set value at most once per scan cycle. Modifications described in [8] can be made to programs that do not satisfy this constraint.

All the constraints can be verified based on the CFG and its associated table. It can be verified that the program corresponding to Tab.I and Tab.III satisfy all the above constraints. We will discuss the reason why we have these assumptions and constraints in the following sections.

C. Cyclic Behavior, Relays and Time

According to [11], there are four ways to model scan cycle: models without scan cycle, models with explicit scan cycle, models with implicit scan cycle, and models abstracted from scan cycle. Our model belongs to the third kind. Models with implicit scan cycle can be simply understood as the modeling of cyclical behavior, but the duration of each scan cycle is not considered. The reasons why we chose this are as follows:

- In practice, the duration of each scan cycle and the execution time of each instruction are not of great importance. What we concern is only the summation of durations of several adjacent scan cycles.
- We want to get a simpler model. If we take into account the execution time of each instruction, several auxiliary notations need to be added into the model. Though it could make our model more precise, it makes the model more complex and the reasoning more difficult which is not necessary.

In PLC, relays are used to store input, internal and output values. The modeling of scan cycle is represented by the modeling of relays. We define a variable $Cycle$ as the total number of scan cycles from the start of the program.

```
Definition Cycle := nat.
```

After the definition of scan cycle, there are three things to do: 1) define the value of each relay at each scan cycle; 2) attach time to each scan cycle; 3) determine the interpretation location of the value and time for each scan cycle.

1) Values of Relays: The values (i.e. ON and OFF) of relays change according to the cycles. In other words, they are functions from $Cycle$ to Boolean:

```
Definition Var := Cycle -> bool.
```

Since Coq is a system based on intuitive logic, the results of logic computation are in the sort $Prop$. In order to facilitate the proof process, we use the following definition instead:

```
Definition Var := Cycle -> Prop.
```

For example, given a relay r of type Var and a cycle i, $r(pred\ i)$ and $(r\ i)$ are used to denote the values of relay r at $(i-1)$-th cycle and i-th cycle, respectively.

2) Time: Time is defined by natural number:

```
Definition Time := nat.
```

Function f associates each scan cycle with a time:

```
Variable f : Cycle -> Time.
```

Function f should satisfy the monotonic property which means that the time attached to scan cycles increases strictly:

```
Hypothesis f_monotonic:forall c, f c < f (S c).
```

3) Interpretations of Values and Times: The value of a relay and time at a cycle can have several interpretations. For instance, given a relay r, the values of r along the execution of PLC program form an infinite trace which is illustrated in Fig.4. This trace alternates the I/O phase and calculation phase infinitely. The inner values of r during the calculation phase is not of concern here. Intuitively, in Fig.4 the first point represents the value of r at the beginning of the I/O phase of cycle 0, the second point represents the value of r at the beginning of the calculation phase of cycle 0, and so on. Thus, $(r\ 0)$ can have at least three different interpretations depending on whether the value is sampled at a, b or c. If b is chosen, abstract trace A is extracted from the concrete trace. In case of c, abstract trace B is obtained. An abstract trace is a division of the concrete trace. Different abstract traces

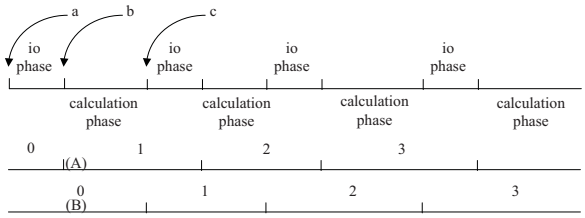

Figure 4. Different interpretations of value at a cycle

separate the I/O phases and calculation phases differently. For trace B, an I/O phase and its following calculation phase belong to the same abstract cycle which is similar to the standard understanding of scan cycle. For trace A, a calculation phase and its following I/O phase belong to the same abstract cycle. Based on different abstract trace, different abstract models can be obtained. For example, two models can be built for the first rung in Fig.2:

```
% abstract model A
Variables i0 i1 m1 : Var.
Hypothesis h_m1 : forall c, m1 c =
 ((i0 (pred c)\/m1 (pred c))/\~i1
(pred c)).

% abstract model B
Variables i0 i1 m1 : Var.
Hypothesis h_m1 : forall c, m1 c =
 ((i0 c \/ m1 (pred c)) /\ ~ i1 c).
```

In the first model, since the calculation of m_1's value at cycle c is based on the values of i_0 and i_1 at the previous cycle, i_0 $(pred\ c)$ and i_1 $(pred\ c)$ are used. In the second model, the calculation of m_1 uses the values of i_0 and i_1 at the current cycle, so i_0 c and i_1 c are used.

The discussion about the interpretations of time is similar to that of relay values. This will be discussed in detail in section IV-E.

D. Rungs and Their Execution Order

Every rung is modeled by a hypothesis in Coq. The execution order of instructions in a PLC program should be reflected by the model. According to the cyclic behavior of PLC and the execution order of instructions, for each node i, its ref_i can be divided into two disjoint subsets: ref_i^c and ref_i^p, where ref_i^c stands for the set of variables whose values at the current scan cycle are used for the execution of i and ref_i^p stands for the set of variables whose values of the previous scan cycle are used for the execution of i. Let us take the sixth rung in Fig.2 for example, $ref_6^c = \{m_1, m_2, m_3, m_4\}$ and $ref_6^p = \{m_5\}$. For any cycle c, $v^c \in ref_6^c$ and $v^p \in ref_6^p$, $(v^c\ c)$ and $(v^p\ (pred\ c))$ are used to calculate m_5. Hence, we have the following Coq codes for the sixth rung:

```
Hypothesis h_m5 : forall c, m5 c =
 (((m2 c\/m3 c\/m4 c)\/m5 (pred c))/\m1 c).
```

E. TON-Timer

As mentioned in section IV-C, based on different abstractions we have different models for TON-timers. We first give the assumption about the preset time of t_1, then choose an interpretation location, finally build a model for t_1.

1) Assumptions about Time: Based on the above notations, definitions of timer bit and preset time of the TON-timer are given as follows (t_1 is used to denote the timer bit and t_1_PT of type $Time$ denotes the preset time):

```
Variable t1:Var.
Variable t1_PT:Time.
```

In practice, the preset time of t_1 must be greater than time span of any adjacent scan cycles:

```
Hypothesis f_TLTCI :
 forall c, f (S c) - f c < t1_PT.
```

2) The Interpretation Location: The concrete trace of quiz machine is shown in Fig.5. We concern the inner values in the calculation phase here. We have several different interpretation locations. Here we choose location a. The abstract trace is the line below. The model of TON-timer t_1 is represented by three axioms:

```
Axiom h_t1_reset : forall c, ~ m1 c -> ~ t1 c.
Axiom h_t1_set :
 forall c1 c2, t1_PT<=f(pred c2)-f(pred c1)->
 being_true m1 c1 c2 -> t1 c2.
Axiom h_t1_true :
 forall c2, t1 c2->exists c1,
 t1_PT<=f(pred c2) - f (pred c1) /\
 being_true m1 c1 c2.
```

Axiom h_t1_reset reflects the second characteristic of TON-timer.

The references to the same cycle c in "m_1 c" and "t_1 c" show the fact that the value of m_1 affects the value of t_1 in the same cycle. The meaning of the second axiom can be explained using Fig.5. Suppose in the abstract trace the time span from $(pred\ c_1)$ to $(pred\ c_2)$ is greater than or equal to t_1_PT (which is presented by "$t_1\_PT <= f\ (pred\ c_2) - f\ (pred\ c_1)$") and m_1 is ON between c_1 and c_2 (which is expressed by the predicate "$being\_true\ m_1\ c_1\ c_2$"). Hence in the concrete trace the time span between a and e is greater than or equal to t_1_PT. Because of the fifth constraint mentioned in section IV-B, the time span between a and b is equal to the time span between e and f. Finally we have that the time span between b and f is greater than or equal to t_1_PT. Note that $rung_2$ contains the timer instruction. Together with the fact that m_1 stays ON from b to g, we have that t_1 is ON at g in the concrete trace, in other words t_1 is ON at cycle $c2$ in the abstract trace, which is the conclusion of the second axiom. The third axiom can be understood in a similar manner.

These axioms forms a parameterized module for a TON-timer. The module has two parameters: IN (which is m_1 in the example) and PT (which is t_1_PT in the example).

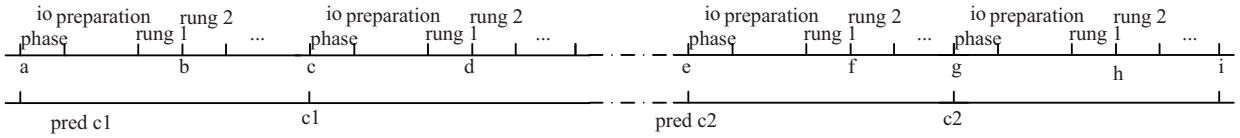

Figure 5. Concrete and abstract traces

In general, the statement of these hypothesis have to be universally quantified over arbitrary inputs and outputs. For the convenience of this study, we assume that the hypothesis have been instantiated by the concrete variables in the program.

V. The Complete Model of Quiz Machine

The complete model of the quiz program is shown below:

```
Variables i0 i1 i2 i3 i4 : Var.
Variables m1 m2 m3 m4 m5: Var.
Variables o0 o1 o2 o3 : Var.

Variable t1 : Var.
Variables t1_PT : Time.
Hypothesis f_TLTCI :
  forall c, f (S c) -  f c < t1_PT.
Axiom h_t1_reset : forall c, ~ m1 c -> ~ t1 c.
Axiom h_t1_set :
  forall c1 c2, t1_PT<=f(pred c2)-f(pred c1)->
    being_true m1 c1 c2 -> t1 c2.
Axiom h_t1_true :
  forall c2, t1 c2->exists c1,
    t1_PT<=f(pred c2) - f (pred c1) /\
    being_true m1 c1 c2.

Hypothesis h_m1 : forall c, m1 c =
  ((i0 c \/ m1 (pred c)) /\ ~ i1 c).
Hypothesis h_m2 : forall c, m2 c =
  (((~ t1 c /\ i2 c /\ ~ m5 (pred c))
  \/ m2 (pred c)) /\ m1 c).
Hypothesis h_m3 : forall c, m3 c =
  (((~ t1 c /\ i3 c /\ ~ m5 (pred c))
  \/ m3 (pred c)) /\ m1 c).
Hypothesis h_m4 : forall c, m4 c =
  (((~ t1 c /\ i4 c /\ ~ m5 (pred c))
  \/ m4 (pred c)) /\ m1 c).
Hypothesis h_m5 : forall c, m5 c =
  (((m2 c \/ m3 c \/ m4 c) \/
  m5 (pred c)) /\ m1 c).
Hypothesis h_o1 : forall c, o1 c = m2 c.
Hypothesis h_o2 : forall c, o2 c = m3 c.
Hypothesis h_o3 : forall c, o3 c = m4 c.
Hypothesis h_o0 : forall c, o0 c=(t1 c/\~m5 c).
```

The above model can be understood as a module with one parameter "t_1_PT" and a property "f_TLTCI" that the parameter should satisfy. By giving "t_1_PT" a constant that satisfies "f_TLTCI" to the module, an instance can be obtained such that all properties the parameterized module hold are also held by the instance.

A. Formalization of Properties

We proved that three expected behaviors, each of which is described in a theorem, hold in Coq. The first of these properties is delineated in the following theorem. It describes the situation that one player presses his button and then the associated light is turned on. The theorem contains various conditions to make sure that the action can proceed. In order to make the theorem clear, several predicates, such as $reset\_then\_start$ and $just\_time\_out$, are introduced. They will be explained along the description of the theorem.

```
Theorem reset_start_time_i2_o0o1o2o3_f :
  forall c1 c2, reset_then_start c1 c2 ->
    forall c3, just_time_out c2 c3 ->
     forall c, S c2 <= c <= c3 ->
       p1_first_presses (S c2) c ->
         forall c4, c <= c4 ->
           not_reset (S c2) c4 ->
           stay_off o0 (S c2) c4 /\
           off_on o1 (S c2) c c4 /\
           stay_off o2 (S c2) c4 /\
           stay_off o3 (S c2) c4.
```

The formulae before the last arrow present the premises, i.e. the behaviors of the environment (including the actions of the host and players); those after the last arrow describe the conclusions, i.e. the expected behavior of the system. Fig.6 is a graphic representation of the theorem, where the words above the horizontal arrow describe the premises and the words below the arrow describe the conclusions. Predicate "$reset\_then\_start\ c_1\ c_2$" expresses the action sequence of the host: the host presses reset button at cycle c_1 and does not press start button from cycle c_1 to cycle c_2, then he presses start button at cycle $(S\ c_2)$ (i.e. the next cycle of c_2). Predicate "$just\_time\_out\ c_2\ c_3$" means the time span between c_2 and c_3 is less than t_1_PT and that between c_2 and $(S\ c_3)$ is equal or larger than t_1_PT. Predicate "$p1\_first\_presses\ (S\ c_2)\ c$" describes the fact that there exists a cycle c such that between $(S\ c_2)$ and $(pred\ c)$ no one presses his button and at c player$_1$ is the only one that presses his button. Predicate "$not\_reset\ (S\ c_2)\ c_4$" means during $(S\ c_2)$ and c_4 the reset button is not pressed.

The conclusion has four predicates: 1) "$off\_on\ o_1\ (S\ c_2)\ c\ c_4$" means that light$_1$ is off between $S\ c_2$ and $pred\ c$ and light$_1$ is on between c and c_4; 2) "$stay\_off\ o_0\ (S\ c_2)\ c_4$", "$stay\_off\ o_2\ (S\ c_2)\ c_4$" and "$stay\_off\ o_2(S\ c_2)\ c_4$"

describe that between $S\ c_2$ and c_4 light$_0$, light$_1$ and light$_3$

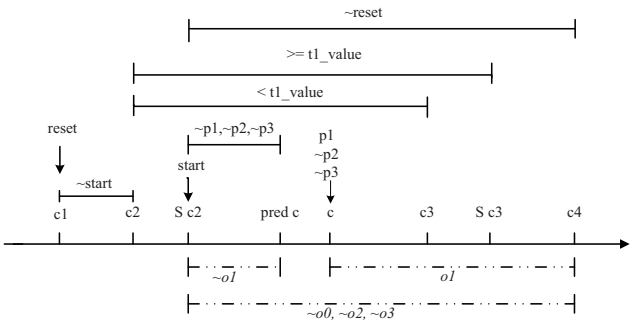

Figure 6. Graphic representation of theorem 1

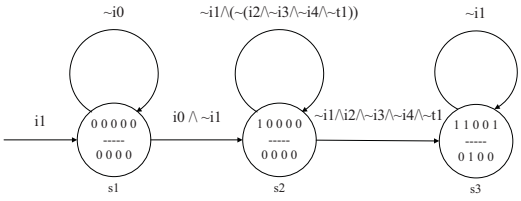

Figure 7. Part of the transition system

are all off.

B. Outline of the Proof

Since the proofs of the three theorems are lengthy – the whole file has more than two thousand lines – we only outline the proof skeleton here.

The model of the quiz machine can be considered as a transition system. The state of the system is an vector consisting of all the internal relays : $(m_1, m_2, m_3, m_4, m_5)$. The inputs of the program (i.e. i_0, i_1, i_2, i_3 and i_4) and the timeout signal (i.e. t_1) are the guards on the transitions. Part of transition system used to prove the theorem is demonstrated in

Fig.7. The numbers in the states above the line indicate the values of m_1, m_2, m_3 and m_5 respectively and those below the line indicate the values of o_0, o_1, o_2 and o_3. The variables above the transitions represent the conditions under which the corresponding transition can take place. Variables not mentioned means the transition does not care the values of these variables. For instance, if the system is at state s_1, then all the output and internal relays are 0. If i_0 is false, whatever values the other variables are the system stays in s_1. If i_0 is true, the system will change to state s_2.

The promises of the theorem express the inputs of the system and their orders:

1) The host presses the reset button.
2) The host presses the start button and does not press the reset button afterwards.
3) Player$_1$ firstly presses his button before the timeout.

We proved that following the above inputs the system

1) reaches s_1 after the host presses the reset button; then
2) it reaches s_2 after the host presses the start button; then
3) it reaches s_3 after player$_1$ presses his button before timeout.

And the outputs of this process coincide with the conclusion of the theorem. The other two theorems are proved in the same manner.

VI. Conclusions

We presented a formalization of the TON timer of programmable logic controllers (PLCs) in the theorem prover Coq. In order to ease the modeling and verification process a sound abstract model is proposed. Based on different interpretations, different abstract models can be obtained. The behavior of the TON-timer is described by a set of axioms at an abstract level, which proves to be appropriate for the formal reasoning in this paper. A quiz machine program with TON-timer is employed as an illustration example throughout the paper. We proved that the PLC quiz machine program works as expected. This work demonstrates the complexity of formal timer modeling.

References

[1] Mader, A., Wupper, H.: Timed automaton models for simple programmable logic controllers. In: Proceedings of the Euromicro Conference on Real-Time Systems, IEEE Computer Society (1999) 114–122

[2] Coq Proof Assistant: http://www.lix.polytechnique.fr/coq/

[3] L'Her, D., Parc, P.L., Marcé, L.: Proving sequential function chart programs using automata. In: WIA '98: Revised Papers from the Third International Workshop on Automata Implementation, London, UK, Springer-Verlag (1999) 149–163

[4] Dierks, H.: PLC-automata: a new class of implementable real-time automata. Theoretical Computer Science **253**(1) (2001) 61–93

[5] Bauer, N.: übersetzung von steuerungsprogrammen in formale modelle. Master's thesis, University of Dortmund (1998)

[6] Moon, I.: Modelling programmable logic controllers for logic verification. IEEE Control Systems Magazine **14**(2) (1994) 53–59

[7] Krämer, B.J., Völker, N.: A highly dependable computing architecture for safety-critical control applications. Real-Time Systems **13**(3) (1997) 237–251

[8] Jiménez-Fraustro, F., Rutten, E.: A synchronous model of iec 61131 PLC languages in signal. In: ECRTS '01: Proceedings of the 13th Euromicro Conference on Real-Time Systems, Washington, DC, USA, IEEE Computer Society (2001) 135

[9] IEC International Standard 1131-3: Programmable Controllers, Part 3: Programming Languages.

(1993)

[10] Siemens: S7-200 Programmable Controller System Manual. Siemens (2003)

[11] Mader, A.: A classification of PLC models and applications. In: In WODES 2000: 5th Workshop on Discrete Event Systems, Kluwer Academic Publishers (2000) 21–23