2011 International Conference on Computer Science and Information Technology (ICCSIT 2011)
IPCSIT vol. 51 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V51.72

The Modelling and Verification of PLC Program Based on Interactive Theorem Proving Tool COQ

Litian Xiao[a,b,+], Mengyuan Li[b], Ming Gu[a] and Jiaguang Sun[a]

[a]National Laboratory for Information Science and Technology, Key Laboratory for Information System Security, Ministry of Education, Department of Computer Science and Technology, School of Software，Tsinghua University, Beijing 100084, China.

[b]Beijing Special Engineering Design and Research Institute, Beijing 100028, China.

Abstract. COQ is an interactive theorem proving tool. The paper abstractly describes the feature of COQ, the architecture and working modes of PLC program with the example of typical PLC. It also introduces the first-order logic syntax and semantics of Intuitionistic Logic. It briefly introduces the main Gallina language syntax elements, the corresponding use methods and main theorem proving tactic on COQ. The work has modeled kernel data type and basic statements and and the denotational semantics of PLC program with Gallina. It has given the correctness proof of PLC program based on theorem proving, i.e. based on semantics function the relationship of configuration between the before codes execution and the after is proved. The main purpose is to prove whether a PLC program satisfies certain nature within a scan period.

Keywords: PLC Program, Embedded System, Verification, Denotational Semantics, Theorem Proving, COQ

1. Introduction

With the increasing requirements for embedded applications, the complexity and the scale of PLC programs is also increasing and becoming larger. As the applications of embedded system in a wide range of security field, relevant to its correctness guarantee becomes more urgent. In terms of PLC, the hardware logic of its foundation has high reliability, but its software correctness is much difficultly guaranteed due to the function complexity of PLC program.

In order to eliminate bugs as possible and verify the correctness of a program, the main methods are test [1]-[3] and formal verification. Although the testing technology is an effective method to find out bugs, it has incompleteness. Finite test cases are difficultly utilized to cover all the accessible codes of a program. Thus, the researchers try to use the formal methods to verify the correctness of program.

One type of formal verification method is model checking which to be verified by state space search based on specified features (such as safety and activeness etc.) upon the abstract model. This method is reliable and complete. However, it faces the biggest problem of “state space explosion” [4]-[6]. The other is the formal verification method of theorem proving. It firstly describes the characteristics of system behaviors with a series of logical formulae. Then it proves the specified objectives correct or falsification by the means of deductive methods with inference rules provided by the logical system (or proof tools).

The theorem proving is well suitable for the verification of infinite state system. The most of the proof tools (such as PVS [7], COQ [8] and Nqthm [9], etc) provides the basic logic that is high-order logic possessed

[+] Corresponding author.
E-mail address: xlt05@mails.tsinghua.edu.cn

of the function of describing the infinite data structure, i.e., insensitive to the scale of state space. But this type of method has relatively higher requirements to the verification personnel.

For PLC program, its formal verification has its own characteristics:

a) PLC program is used in the most embedded hardware environment, so its logic structure is relative simple, the sentence categories is few in program, its program is relative short.

b) PLC program is written in hardware machine instruction (or ladder diagram). It is convenient for the abstracted process of model, even in some times it can be used directly as models for verification (similar to assembly language, Verilog, VHDL, etc.).

c) Although PLC program is simple, but it still has the most mechanism of high-level programming language.

d) PLC program completes in sequence within a scan period, and then out images are refreshed and next scan period is started. From the interior of the scan period, it has characteristics of sequence program. From the whole scan period, what showed by PLC is the output response to the various input signals. Though its response is not fixed — because the accumulation of many kinds of timers and counters is span-scan period, a PLC program can be simply regarded as the fixed transformation logic of input-output within a scan period.

Compared with the verification of general assembly or high-level language program, PLC programs' has the advantages that are relative small scale, relative simple structure, relative clear control logic and close to mathematical representation.

In order to verify the correctness of PLC program and the temporal sequence feature of it within a scan period, we address the research for verification approach based on COQ which is a theorem proving tool. To using the COQ, PLC program need to be modeled at first. The paper defines the grammatical components of the typical PLC program based on Gallina specification language of COQ for the program modeling, and expresses it with Gallina. Its main idea is the structural induction conversion of the semantics. It deals with the COQ descriptions of PLC program semantics and proof strategies for proving some program natures. Theses work has established the foundation of COQ verifying PLC program.

2. The Features of Interactive Theorem Proving Tool Coq

COQ and some famous theorem proving tools are generally adopted with Intuitionistic Logic [10] for nature reduction. The logic has the corresponding relation between it and program. On the one hand, the concepts of formula, proof and checking in logic correspond to that of type, function and type checking in functional program. On the other hand, the process of writing functional program can be mutually conversed with the process of theorem proving, which is Curry-Howard correspondence or isomorphism [10][11].

The Construction Theory of MartinLof *et al* and is integrated into the extended Automath system by Huet, Cousineau, Coquand and *et al*, i.e. Calculus of Constructions. Also Coquand gives the termination proof of the calculus [12]. COQ is developed by Inria (a research organization in France). It provides an integrated interactive platform with program description or definition and nature proving. It can be used in various industries for the development of trusted program, also be used for the specification and the verification of program [8].

Gallina is the specification language of COQ, which is built-in a number of commonly used programming language syntax and type such as integer (*Z*), natural number (*nat*), boolean variable (*bool*), character (*ascii*), string (string) and the types of list (*list*) and choice (option) etc. derived from the former construction.

In the COQ system, a term and an expression work under certain context and in some scope. The same expression has different meanings sometimes in different scope. After a term or an expression is declared, it immediately has an only type. For example, *Variables X Y: Prop* declares the type *Prop* (i.e. proposition type) of variable *X* and *Y* . When the command *Check X* is used for examining the type corresponded to *X*, COQ gives the response, i.e. *Prop*. Any expressions have the types of them and cannot be deduced whether the expressions corresponded to types are illegal [13].

Inductive method can be user-defined new data type [14]. For example, in the type library of COQ the definition of natural number can be

Inductive nat : Set := | O | S : nat → nat ,

i.e. natural number type has two constructors *O* and *S*. It can be equivalence to the following inductive definition:

O ∈ nat;

If m ∈nat, then S m ∈nat.

Therefore, *O, S O, S (S O)…* are natural numbers.

Two definitions approach of term and function in COQ are often direct definition and inductive definition[12] which the former keyword is *Definition* and the latter keyword is *Fixpoint*. Their difference is that the latter is allowed to recursion calling itself within function body. For example,

```
Fixpoint fac (n: nat) : nat :=
   match n with
   | O ⇒1
    | S m ⇒ n * (fac m)
end.
```

It defines a recursion function *fac* which receives a *nat* type parameter and returns a *nat* type value. In function body, *match* statement matches form to *n*, i.e. if the *n* value is *O* (natural number 0) then the function returns the value *1*, if *n* forms such as *S m* then it returns *n*(fac m)*. Where, there is a recursion calling.

The proved theorem and lemma etc. can be defined by keyword *Theorem* and *Lemma*. The following statements in COQ: Theorem *add_perm: ∀P Q: Prop, P∧Q↔Q∧P.*

is expressed that conjunction operation has exchangeable nature. Logic symbols ¬, ∧, ∨, →, ↔, ∀, ∃ are respectively expressed with ~, ∧, ∨, ->, <->, *forall*, *exists* in COQ.

Besides modeling language, COQ also provides a series of theorem proof tactics such as *intro*, *split*, *unfold*, *simpl* and *reflexivity* for accomplishing interactive theorem proving. A tactics is corresponding to converse application of rules in Intuitionistic Logic. Main proving tactics and other details about COQ can be referred to [8] [12] [15] [16].

3. The Architecture Definition of PLC Program

The cyclic scanning mode is adopted in PLC. When in operation, each scan period is divided into five stages such as internal processing, program communication, input scanning, program performing and output processing. Among the stages internal processing, program communication are auxiliary. The primary units composed in PLC system hardware are input relay unit (*X*), output relay unit (*Y*), auxiliary relay unit (*M*), status register (*S*), timer (*T*), counter (*C*), data register (*D*), PLC pointer (*P*) and so on. The various series of PLC typical instructions are basically similar, including the normal opened/closed contacts connecting to the control bus (I/O: *LD/LDI*), series connection to the normal opened/closed contacts (*AND / ANDI*), parallel connection to the normal opened / closed contacts (*OR/ORI*), block operation (*ANDB/ORB*), negating (*INV*), special element set 1/0 and remaining (*SET/RST*), non-operation (*NOP*), main program end (*END*), stepping (*STL* or *RET*), push stack / read stack / pop stack (*MPS / MRD / MPP*), condition jump (*CJ*), loop (*For…NEXT*), subprogram call and return (*CALL/SRET*), arithmetic operation (*ADD, SUB, MUL, DIV, INC, DEC*), digitwise logical operation (*WAND, WOR, WXOR, NEG*), comparison (*CMP*), data transfer (*MOV*), etc. The detailed instructions can be referred to the related manuals and reference [17].

The exact mathematical description of PLC program architecture can be defined as following:

a) To introduce an argument $V_z \in \{0,1\}$ to every I/O relay unit, aux relay unit and status register *Z*, e.g. the input unit *X0* corresponding to V_{x0}.

b) To introduce a corresponding argument $V_z \in \mathbb{N}$ (the domain of natural numbers) to every data register unit and counter unit *Z*.

c) To regarding the pointer value (such as subprogram jump entry address, conditional jump statement address) as constant value.

d) Not considering the mathematical description of the timer due to the verification technology in this paper not related to the real-time problem.

e) The above arguments are divided into two types: global argument and local argument. Among them, the unit operated by *SET/RST* command and the counter unit are global arguments, and the rest are local arguments. *GV* and *LV* are separately denoted as the sets of global and local arguments.

f) In addition, for the counter units, because a scheduled counting value is required and this value will be set for many times. A temporary argument E_C will be set for each counter *C*, which E_C value is effected by

OUT command. For example, *OUT C0 K200*, when E_C is assigned as 200.

g) Due to stack operation exiting, a structure $S_M \in \{0,1\}^*$ used to record the current changes of bus stack. To consider the following codes,

```
LD     X0
MPS
  AND     X1
  OUT     Y1
MRD
AND     X2
MPS
AND     X3
MPP
```

Supposed $V_{X0}=1$, $V_{X1}=1$, $V_{X2}=0$, $V_{X3}=1$, then after implementing the codes stack values are respectively ε,1, 1, 1, 1, 1, 1·0, 1·0, 1. where empty string ε is expressed that stack is empty.

h) There are a large number of commands similar to *AND/OR/ANDB/ORB/ANDI/ORI*, which results of the implementation rely on the current bus connecting status. Meanwhile, the implementation of these commands can change the current bus status, as well as the block series and parallel commands rely on bus connection status prior to some steps. Therefore, it is necessary to set a stack for the current bus connecting status.

Generally V is expressed as an argument and C is constant, which may be with subscript. A quad $<\omega_M, \omega_T, \sigma_G, \sigma_L>$ is expressed as the current Configuration. Where, ω_M is bus stack and ω_T is current connecting stack, σ_G is assignment functions of global variables and σ_L is local, i.e. the current value of every global variable V is assigned as $\sigma_G(V)$ and the value of every local variable V is $\sigma_L(V)$.

In order to give the denotational semantics of PLC program, BNF paradigm is introduced as following:

$$\begin{aligned} e ::= {} & V \\ & \mid C \\ & \mid e+e \mid e-e \mid e\times e \mid e\div e \\ & \mid e\,□\,e \mid e\,□\,e \mid e\oplus e \mid \neg e \\ & \mid e\&e \mid e|e \mid e\hat{\ }e \mid \bar{e}\ . \end{aligned}$$

Where +, -, × and ÷ are corresponding arithmetic operation symbols of "plus, minus, multiplication, division". Symbol □, □, ⊕ and ¬ are logic operation of "and, or, XOR, not". Symbol &, |, ^ and ¯ are bit operation of "bit and, bit or, bit XOR, bit not".

Given configuration $CF=<\omega_M, \omega_T, \sigma_G, \sigma_L>$, the semantics value under an expression e is denoted by $[\![e]\!]_{CF}$. The inductive definition is as follows.

- $[\![C]\!]_{CF} = C$.
- $[\![V]\!]_{CF} = \begin{cases}\sigma_G(V), V\in GV \\ \sigma_L(V), V\in LV\end{cases}$
- $[\![e_1 * e_2]\!]_{CF} = [\![e_1]\!]_{CF} * [\![e_2]\!]_{CF}$, where $* \in \{+,-,\times,\div,□,□,\oplus,\&,|,\hat{\ }\}$, its operation definition is same to normal.
- $[\![!e]\!]_{CF} = ![\![e]\!]_{CF}$, where $! \in \{\neg, \bar{\ }\}$, its operation definition is same to normal.

4. PLC Program Modeling Based on COQ

In PLC program the components need to be divided into two categories: one category is the component of storage one bit such as input relay unit (*X*), another is that of storage data which is 16 or 32 bit such as counter (*C*). Therefore, σ_G and σ_L need to refine further which is separated into the different functions composed of "bit type" and "data type" registers.

The following structure data type can model the configuration.

```
Record config := MkConfig {
                    M_Stack : list bool;
                    T_Stack : list bool;
                  L_Bit_Map : string → bool;
                   L_Data_Map : string →Z;
                G_Bit_Map : string → bool;
                G_Data_Map : string → Z
```

}.

Where, *Record* is the keyword of COQ which its data type is equivalence to the structure type of high-level language. Local arguments assigned function σ_L is divided into two fields: *L_Bit_Map* and *L_Data_Map*. Global arguments assigned function σ_G is divided into two fields: *G_Bit_Map* and *G_Data_Map*.

The COQ description of PLC program statements can be defined nil, unitary, binary and ternary operations of PLC program with type declaration statement.

```
Inductive PLC_Z_OP:=| ORB | ANDB | INV | NOP | MPS | MRD | MPP.
Inductive PLC_U_OP:=| LD | LDI | AND | ANDI | OR | ORI | OUT | SET | RST | INC | DEC | NEG.
Inductive PLC _B_OP:=| OUTC | MOV| XCH.
Inductive PLC_T_OP:=| ADD | SUB | MUL | DIV | WAND | WOR | WXOR| CMP.
```

Where, *PLC_Z_OP*, *PLC_U_OP*, *PLC_B_OP* and *PLC_T_OP* separately describe the commands of which don't need any operands, need an operand or two operands and three operands.

In PLC control circuit, also an only quaternary operation is *ZCP* statement. So, type declaration is

```
Inductive PLC_F_OP := ZCP.
```

The following declaration statements is defined as main PLC program statements.

```
Inductive  PLC_Sent := | Z_Cmd: PLC_Z_OP→ PLC_Sent
|U_Cmd: PLC_U_OP→ string→ PLC_Sent
|B_Cmd : PLC_B_OP→ string→string→PLC_Sent
|T_Cmd : PLC_T_OP→string→string→string→PLC_Sent
|F_Cmd :PLC_F_OP→string→string→string→string→PLC_Sent
|Conseq: PLC_Sent→ PLC_Sent→ PLC_Sent
|Cond : PLC_Sent → PLC_Sent → PLC_Sent
|Loop_Z : string → PLC_Sent →PLC_Sent
|Loop_C : nat → PLC_Sent→ PLC_Sent.
```

Where, *Z_Cmd op*, *U_Cmd op X*, *B_Cmd op X Y*, *T_Cmd op X Y Z* and *F_Cmd op X Y Z W* are separately corresponding to nil, unitary, binary and ternary operation commands. *Conseq*, *Cond* and *Loop* constructor are separately constructed synthetic sequence, conditional branch and loop commands.

For example, considering the PLC program,

```
        LD    X1
        JC    L
        AND  X2
  L:    AND        X3
       OUT   Y1
```

in COQ the program can be expressed as

```
Conseq ( Conseq  (U_Cmd   LD "X1")
(Cond  (U_Cmd  AND "X2")  (U_Cmd  AND  "X3"))
(U_Cmd  OUT   "Y1") ).
```

Based on formal denotational semantics, the denotational semantics of PLC program is a mapping from a configuration to another. The declaration is described as followings.

```
Fixpoint Prog_Semantics (St : PLC_Sent) (C: config): config :=match St with
| Z_Cmd  op ⇒ (Z_Cmd_Semantics C op)
| U_Cmd  op  X ⇒ (U_Cmd_Semantics C op X)
| B_Cmd  op X Y ⇒  (B_Cmd_Semantics C op X Y)
| T_Cmd  op X Y Z ⇒ (T_Cmd_Semantics C op X Y Z)
| F_Cmd op X Y Z W⇒ (F_Cmd_Semantics C op X Y Z W)
|Conseq St′ St″⇒ Prog_Semantics (St″(Prog_Semantics St′ C))
| Cond St′ St″ ⇒ match (T_Stack C) with
| 1:: t ⇒  Prog_Semantics St″ C
| _  ⇒  Prog_Semantics (Conseq St′ St″) C
| Loop_Z  X St′ ⇒
   if (Is_glb_var_name X) then
      if  (G_Data_Map C X <> 0) then
       let  C′:= Prog_Semantics St′  C in
      C″:= MkConfig (M_Stack C′) (T_Stack C′) (L_Bit_Map C′)
      (L_Data_Map C′) (G_Bit_Map C′) (Adapt (G_Data_Map C′) X (G_Data_Map X)-1) in
      Prog_Semantics (Loop_Z X St′) C″
    else C
  else
  if (L_Data_Map C X <> 0) then
   let C′:= Prog_Semantics St′ C in
   let C″ :=MkConfig (M_Stack C′) (T_Stack C′) (L_Bit_Map C′)(Adapt (L_Data_Map C′) X
(L_Data_Map C X)-1) (G_Bit_Map C) (G_Data_Map C) in
     Prog_Semantics (Loop_Z X St′) C″
  else C
  | Loop_C K St′ ⇒ if (K <> 0) then let C′:= Prog_Semantics St′C in Prog_Semantics(Loop_C K-1 St′) C′
   else C
  end.
```

Where, *Adapt* accomplishes the single substitution function (i.e. *f{x:=t}*). *string_eq_bool* is used for the function that is compared two strings whether to equal. *prefix* is a function pre-defined in *String* library, which it is used to judge whether a string is the prefix of another string. In *Prog_Semantics* function definition, some assistant functions *Z_Cmd_Semantics*, *U_Cmd_Semantics*, *B_Cmd_Semantics*, *T_Cmd_Semantics* and *F_Cmd_Semantics* is the semantics that calculate nil, unitary, binary, ternary and quaternary commands. Specific definition of these functions can be expanded, which we shall discuss in another paper because of limited length.

5. Nature Proving of PLC Program Based on COQ

Some typical PLC programs will be examples to illustrate how to prove the nature of PLC programs in COQ. At first, the example starts from a basic module which accomplishes simply data exchange. The program is showed as following.

```
MOV D1 D3
MOV D2 D1
MOV D3 D2
```

Its proved nature is described with the first-order logic formula of Intuitionistic Logic:

∀C: config. (L_Data_Map) C "D1" = L_Data_Map (Prog_Semantics Cod C) "D2"
∧ (L_Data_Map) C "D2" = L_Data_Map (Prog_Semantics Cod C) "D1".

Where, Prog_Semantics Cod *C* is the COQ description of above three line codes, i.e.

Conseq ((B_Cmd MOV "D1""D3")
Conseq (B_Cmd MOV"D2""D1") (B_Cmd MOV "D3""D2")).

The formula can be proved with the tactics of *intro*, *split*, *unfold*, *simpl* and *reflexivity*.

Considering that the value is summed from *1* to *n*, $\sum_{i=1}^{n} i$ which corresponding codes is showed as following.

```
FOR D1
ADD D1 D2 D2
NEXT
```

The *Cod′* description in COQ is

LOOP_Z (T_Cmd ADD "D1""D2" "D2").

The assistant function is defined as following for describing accumulation.

```
Fixpoint sum_up (n: nat) : nat :=
      match n with
               | O ⇒ O
               | S m ⇒ n + (sum_up m)
      end.
```

Then proved nature is described with the first-order logic formula:

∀C: config. (L_Data_Map C "D2")=0
→ (L_Data_Map (Prog_Semantics Cod′ C) "D2")
= (sum_up (L_Data_Map C "D1")).

Induction need to use for proving the nature. Inductive object should be *L_Data_Map C "D1"*, i.e. after tactic *intro* the tactic *induction L_Data_Map C "D1"* should be used. Where, the first proving object can be eliminated by tactic *simpl* and *reflexivity*. The second proving object can be proved by tactic *rewrite* or *change*, *simpl* and *reflexivity*.

6. Conclusion and Further Work

With the example of typical PLC, the paper abstractly describes the feature of COQ, the architecture and working modes of PLC program. It also introduces the first-order logic syntax and semantics of Intuitionistic Logic. Further with theorem proving tool COQ, it briefly introduces the main Gallina language syntax elements, the corresponding use methods and main theorem proving tactic.

The work has modeled kernel data type and basic statements and and the denotational semantics of PLC program with Gallina. It has given the correctness proof of PLC program based on theorem proving, i.e. based on function Prog_Semantics the relationship of configuration between the before codes execution and the after is proved. The main purpose is to prove that a program should satisfy certain nature within a scan period. It is not much on the related work about correctness verification of PLC program based on theorem proving, which there mainly are [18] and [19].

It is the foundation of verification and theorem proving for PLC program. Given the denotational semantics of PLC program, we have further studied that the semantics is transformed into other mathematical model, i.e. the transition system (model) need to use in model checking technology. Specificially, the research further develops PLC program modeling relation based on Gallina language of COQ platform and makes the COQ description for each PLC syntax elements automaticly defined. Then the development of a definition tool can really help PLC program verifcation.

7. Acknowledgements

This research is sponsored by NSFC Program (No.90718039 & No.91018015) and 973 Program (No.2010CB328003) of China.

8. References

[1] H. Zipori, *et al*. Approaches and implementation of software test and development system for embedded computer systems, in *Embedded System Design*, vol.35(6), pp.30-39, 2005.

[2] E. A. Lee. Computing for embedded systems, in *Proceeding of IEEE Instrumentation and Measurement Technology Conference*, Budapest: Hungary, 2001, pp.100-105.

[3] B. Kang, *et al*. A Design and Test Technique for Embedded Software, in *Proceedings of the 2005 Third ACIS Int'l Conference on Software Engineering Research, Management and Applications*, 2005.

[4] E.M. Clarke, E.A. Emerson. Design and Synthesis of Synchronization SkeletonsUsing Branching Time Temporal Logic. D. Kozen, (Editor) *Logic of Programs*. Springer-Verlag, 1982, vol. 131 of Lecture Notes in Computer Science, 52-71.

[5] J.P. Queille, J. Sifakis. Specification and Verification of Concurrent Systems, in *International Symposium on Programming 1982*. Springer-Verlag, 1982, vol. 137 of Lecture Notes in Computer Science, 216-230.

[6] L. Lamport. Proving the Correctness of Multiprocess Programs. *IEEE Trans on Software Engineering*. 1977.

[7] S. Owre, S. P. Rajan. J. M. Rushby, N. Shankar, M. K. Srivas. PVS: combining specifications, proof checking and model checking. In R. Alur and T. A. Henzinger (Eds) CAV'96, Vol 1102 of *LNCS*, pp 411-414. 1996.

[8] The Coq toolkit. *http://*coq.inria.fr/.

[9] R. S. Boyer, J. S. Moore. Proving theorems about lisp functions. Journal of the ACM, 22(1): 129-144. 1975.

[10] W. A. Howard. The formulae-as-types notion of construction. In J.P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479-490. Academic Press, 1980.

[11] J-Y. Girard, Y. Lafont, P. Taylor. *Proofs and types*. Cambridge University Press, 1989.

[12] Yves, Pierre Casteran. *Interactive Theorem Proving and Program Development: Coq' Art: The Calculas of Inductive Constructions*.Springer-Verlag, 2009.

[13] H. Barendregt. Introduction to generalized type systems. Journal of Functional Programming. 1(2): 125-154, 1991.

[14] P. Aczel. An introduction to inductive definitions. In J. Varwise (Ed), *Handbook of Methematical Logic*, Vol 90 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1977.

[15] C. Paulin-Mohring. Inductive definitions in the system Coq – rules and properties. In M. Bezem and J.-F. Croote, editors, *Proceedings of the conference Typed Lanbda Calculi and Applications*, vol 664 of LNCS, Springer-Verlag, 1993.

[16] D. Scott. Constructive validity. In Proceedings of Symposium on Automatic Demonstration, Vol 125 of LNM , pp. 237-275. Springer-Verlag, 1970.

[17] Karl-Heinz John, Michael Tiegelkamp. IEC 61131-3: Programming Industrial Automation Systems. Springer-Verlag, Berlin Heidelberg, 2001.

[18] W. Rui. Modeling and Verification of Program Logic Controllers with Timed Automata in *IET Software*. 2004.

[19] Chen Gang, Song Xiaoyu, Gu Ming.PLC Program Verification and Analysis Using the COQ Theorem Prover. Peking University Journal (Natural Science Edition), Vol 46, No. 1, pp: 30-34, 2010.