Safe programming of PLC using formal verification methods

O. De Smet[1,2], S. Couffin[1], O. Rossi[1], G. Canet[3,4], J.-J. Lesage[1], Ph. Schnoebelen[3], H. Papini[4]

[1] Laboratoire Universitaire de Recherche en Production Automatisée.
Ecole Normale Supérieure
61 avenue du Pdt Wilson
94235 Cachan Cedex
France
Fax: 33 1 47 40 22 20
{de_smet, couffin, rossi, lesage}@lurpa.ens-cachan.fr

[2] Chaire de Fabrications Mécaniques.
CNAM Paris
2, rue Conté
75003 Paris
France

[3] Laboratoire Spécification et Vérification.
Ecole Normale Supérieure
61 avenue du Pdt Wilson
94235 Cachan Cedex
France
Fax: 33 1 47 40 24 64
{canet, phs}@lsv.ens-cachan.fr

[4] Corporate Research Center ALCATEL.
Route de Nozay
91461 Marcoussis Cedex
France
Fax: 33 1 69 63 17 89
helene.papini@alcatel.fr

ABSTRACT

The Corporate Research Center of ALCATEL (Marcoussis, France) and two laboratories of the Ecole Normale Supérieure (Cachan, France) have been involved since March 1998 in a research project in the field of formal verification of Programmable Logic Controllers (PLC) programs. The Sequential Function Chart (SFC), Ladder Diagram (LD) and Structured Text (ST) languages have been chosen among the five languages defined in the IEC 61131-3 standard. The aim of this project is the development of methods and algorithms enabling the designer to decide if the PLC program verifies the safety, liveness and time properties specified in the requirements.

SAFE PROGRAMMING

Industrial Programmable Logic Controllers (PLC) are used extensively in manufacturing industries for complex control applications. In today’s economic context, the design of these control applications is of a great impact in terms of productivity and production costs. Because of those costs, of the complexity of the control systems and of the multiple hardware/software combinations (many programming tools,…), the designer has to take the safety of these systems into account. In this context it is necessary to provide the designer with verification methods that ensure the safety and liveness of the control system.

One way is to ensure the safety of PLC programs by using safety “framework” while programming. For example the IEC 61131 standard improves the reusability and the quality of the programs thanks to PLC manufacturer-independent modular and structuring languages. Nevertheless, having common elements for all programming languages is not sufficient to ensure the quality of the programs and this standard lacks practical verification methods.

The other way is to verifying the written programs. This is the main goal of this paper.

THE IEC 61131-3 STANDARD

The IEC 61131-3 standard [1] [2] defines the syntax and, for a lesser part, the semantics of four programming languages for PLC, as well as a structuring one (Sequential Function Chart).

The four programming languages are Ladder Diagram, Function Block Diagram (graphic programming languages), Structured Text and Instruction List (textual ones). The Ladder Diagram language (LD), based on relay ladder logic diagrams, permit the description of boolean functions. In the Function Block Diagram language (FBD), the programming features are represented as graphic blocks. The Structured Text language (ST), close to Pascal, allows procedural, conditional and loop statements. The Instruction List is an assembly code-like language.

The fifth language is defined in order to structure the internal organisation of PLC programs or function blocks. This graphical language called SFC (Sequential Function Chart) is an extended state machine that contains primitives to describe sequential, parallel and alternative behaviours. It enables the partitioning of a PLC program (or function block) into a set of steps and transitions interconnected by directed links. Each step is associated to a set of actions and each transition is associated to a transition condition.

One of the aims of IEC 61131-3 is to provide PLC programming designers with a modular and structuring tool that will improve the reusability [3] and the quality [4] of the programs, especially in the sense of reliability and safety. Nevertheless, having common elements for all the programming languages is not enough to ensure the quality of the programs and the standard lacks practical verification methods.

PLC PROGRAMS VERIFICATION METHOD

Research on verification methods is a quite recent activity in the Automation field while many works have been performed and successfully tested in Computer Science. Moreover PLC programs verification is close to programming languages verification. In both fields, properties to be verified concern safety, reliability as well as system availability [5]. For these reasons, most of the PLC verifications approaches [6] (including ours) rely on principles close to those used in Computer Science [7], [8].

The method principle

Our method relies on two main phases:

- The behaviour of the PLC programs is modelled as transition systems synchronised by message exchanges. This modelling consists in the definition of the operational semantics. It permits the clear and non-ambiguous definition of PLC program behaviour.
- The implementation/coding in a model checking tool to verify safety and liveness properties. It is based on the operational semantics.

MODELLING USING SYNCHRONISED TRANSITION SYSTEMS

Operational semantics modelling

For each language used in the PLC program, two steps have to be done:

- definition of the semantics based on the IEC 61131-3
- definition of transitions systems representing the elements of the language, and for the ST and LD languages, definition of the assembly rules to compose elements as a program. This leads to the definition of an operational semantics.

The operational semantics consists of one transition system modelling the PLC execution cycle, plus for each language one transition system representing the behavioural rules of the chosen language and several transition systems, which models the elements of the program (see in fig. 2 this structure in the case of an SFC program, "!" means sending and "?" means receiving). The modelling of the behavioural rules depends on the chosen language.

For the SFC language, the structure of the SFC is transformed in a transition system which computes the evolution of the program. For the LD and ST languages, the approach is more a compiling of the program in transition systems.

PLC execution cycle

One transition system models the PLC execution cycle, it includes 3 phases (read inputs, compute, write outputs). This modelling is done once for one kind of PLC (for example cyclic or periodic): it does not depend on the studied programs. The compute phase is used to start different sub-programs (fig. 1). Pre-programs written in ST and/or LD are first started. At their completion, the main program in SFC begins. At the end of this one, the post-programs written in ST and/or LD are processed. This finishes the compute phase.

The SFC language is used to structure the PLC program. SFC sub-programs can be called only by another SFC program. The ST and LD languages can be used in post/pre part of the PLC program or used by the SFC part to describe action or transition-condition.

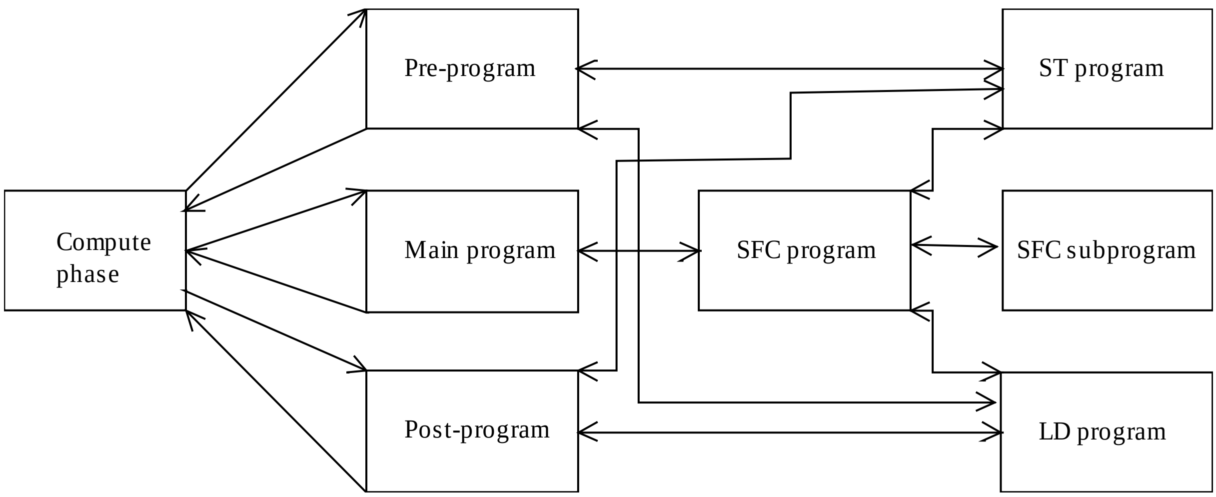

Figure 1: Structure of PLC program: PLC cycle, languages used

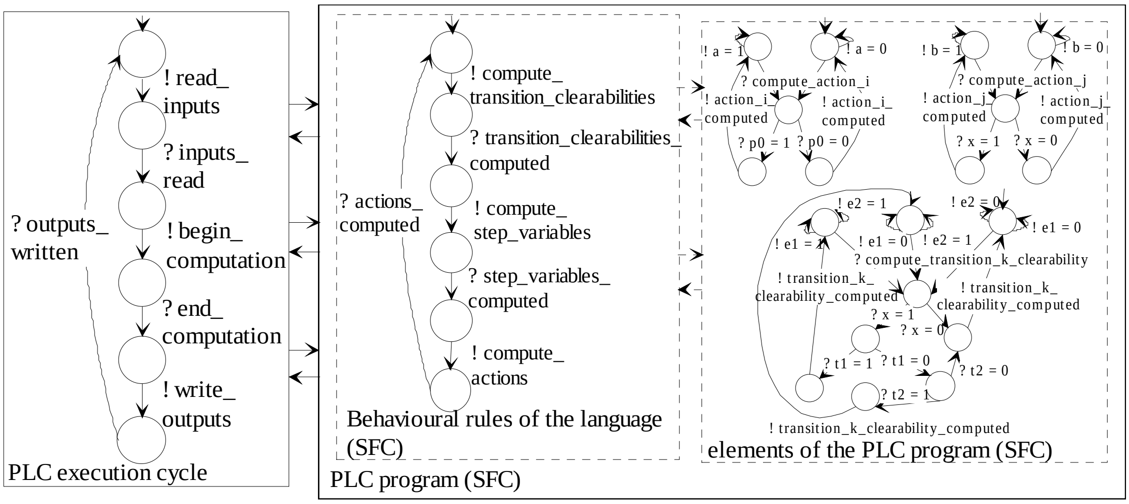

Figure 2: Modelling of the PLC cycle, SFC modelling in the compute phase

SFC language modelling

For SFC, the operational semantics in transition systems (fig. 2) models how it computes (for example the step variables are computed after the transition clearability computation) the evolution of the program with regards to the evolution rules. The program is modelled by several transition systems: one for each step variable, one for each transition to model the clearability, one for each action,…. The step and transition modelling is strongly influenced by the syntax (parallel sequences, sequence selections,…)

LD language modelling

Ladder Diagram language is based on relay ladder logic diagrams and enables the description of boolean functions. Furthermore, as it is a low level design language, it is difficult to write correct LD programs. Therefore, the need of a verification tool is important.

LD programs are usually represented with graphic components. In order to ease LD programs parsing, a textual form for LD is defined in [9]. After the definition of the syntax, a mathematical interpretation for LD primitives should be defined. A formal description of the behaviour of LD programs is given.

To model a LD program behaviour, the interpretation of all the LD primitives has to be mathematically defined. Contacts allow the assignment of a value to a boolean variable. The boolean value held by the wire between the two parts of the rung depends upon the variables of the contacts and the combination instruction between contacts.

When the interpretation of each primitive of the language has been mathematically defined, the behaviour of a whole LD program can be described using a transition system. A simple way to build a behaviour state transition system is to model the internal dynamics of the program i.e. the detailed execution of the program. This internal transition system is naturally defined from LD execution rules, the LD primitives interpretation and the cyclic PLC execution pattern.

In the detailed internal transition system, states are labelled with the value of all the program variables, plus *lc*, a control variable (fig. 3). *lc* enables to store the current step of the execution pattern. Each transition models either a rung evaluation or the start of the program. Rungs are evaluated sequentially, unless a jump instruction is specified in a rung. The “end of cycle” state represent the end of the program.

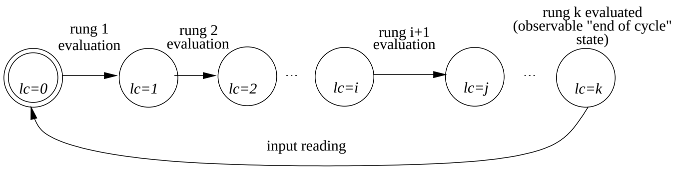

Figure 3: LD transition system cycle

ST language modelling

Structured Text language is similar to Pascal computer-programming language and enables the description of procedural computation. Those procedures can be used as actions or transition-condition for SFC programs. They can be also used as autonomous program for pre/post PLC programs or for the description of the function blocks behaviour.

For the ST language, another approach is used. After defining a formal grammar of the language, each element of the language can be described by a simple automata. Elements can be grouped to form a statement which yelled a boolean value. For the procedural part of the language, statements can be assembled to build sequences which generally just terminate. All the rules used to assemble elements (statements) in a statement (sequence) in the grammar. To assemble statements as a sequence, we use composition and reduction rules to expand transition systems (fig. 4). The process of building the transition system for a statement is a recursive one.

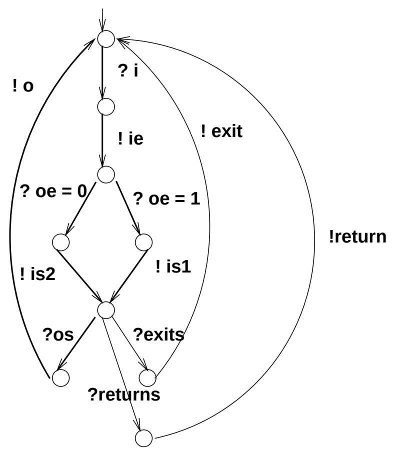

Figure 4: Composition reduction rule for "if-then-else" primitive

Multi-languages modelling

For each language, the verification model has been validated independently: the SFC language [10], the LD language [9], [11].Work for the ST language is more recent, former work has been done for the IL language [12] and is extended for ST.

For a multi-languages PLC program, we must ensure:

- the transition systems modelling the program in a language should be independent of the PLC cycle;
- one transition system modelling a part of the PLC program in a language can be called by another which describes another part in a possibly different language.

To do that, the PLC execution cycle must be clearly isolated and used to separate the reading, computing, and writing phases (as shown in fig. 2 for the SFC language).

CODING IN A MODEL-CHECKING TOOL

Among the Computer Science formal verification methods, both structural and sequential equivalence checking, model-checking and formal proof reasoning [13]), we have chosen temporal model checking.

Model checking is a technique in which a finite model of the system (a transition system) is built and the expected properties (specification of behaviour) of the system are checked on this model. The representation of this transition system can either be explicit (the complete transition system is built) or implicit (symbolic representation of the transition system). In the temporal model checking, the system is modelled as a finite state transition system and the properties are expressed in a temporal logic. A search procedure (exhaustive state space search or reachability analysis) is then used to check whether the expected properties are verified on the finite state transition system.

The model checker we choose is Cadence-SMV (Symbolic Model Verifier) [14] developed by the Carnegie Mellon University. SMV implicitly represents (using BDD) the state automaton modelling all possible program states. Then it checks whether properties expressed by the programmer are satisfied or not. If they are not, it exhibits a sequence of states leading to the contradiction of the assertion. Methods to verify PLC programs using SMV can be found in [9] for the Ladder Diagram language and [10] for the Sequential Function Chart of the IEC 61131-3 standard.

Method for multi-languages programs

All the transition systems defined in the operational semantics are coded in the SMV language. The following paragraphs explains how it is done for each language.

Method for the SFC elements

The SFC program behaviour has to be modelled in finite states to be coded in SMV. The states of the chosen transition system contain all the inputs and outputs of the system, as well as all the step variables of the SFC. The transitions express all the possible evolutions between states. In practice, transition computation includes the evolution rules of an SFC depending on its inputs. As the SMV language enables to define the transitions by giving the NEXT value of the variables, we have chosen to use the well-known mathematical modelling of the SFC defining the new value of a step variable from the previous value of step variables and from the inputs of the SFC. This modelling is described in [10].

Method for the LD elements

Once the behaviour of a LD program has been modelled using a state transition system, it is necessary to encode it into a model-checker to perform automatic verification of properties. Because Cadence-SMV handles formal expressions, and has enough expressiveness, this implementation is made without semantic loss [9], [11].

Method for the ST elements

For ST programs, the same method is applied to code the transition system in Cadence-SMV language.

TEMPORAL PROPERTIES VERIFICATION

Once the state transition system that corresponds to a PLC program has been suitably described for Cadence-SMV, it is possible to automatically check behavioural properties expressed in linear temporal logic (LTL) [15]. This temporal logic is tailored for linking individual properties in complex temporal (i.e., "before-and-after") ways, where boolean combinations are allowed.

The global behaviour of the program must be checked only when outputs latch. The properties should be modified to hold only when the transition system of the PLC program reaches this state. The properties at this stage are generally on the behaviour of the system. A liveness property can be verified when the output state is reached. Safety properties can be written with same modification.

A short example

The following system has been treated with our method. It deals with the control of a tool-holder turret (fig. 5) of a turning-centre. The tool-holder turret is for twelve place (for live or fixed tools), it can rotate clockwise (CW) or counter-clockwise (CCW). The control has to minimise the time for tool-changing.

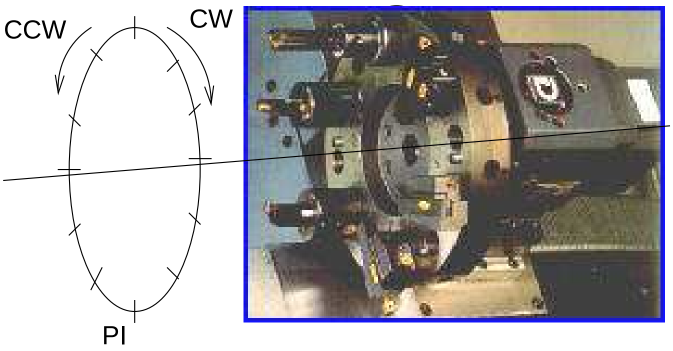

Figure 5: turning-centre tool-holder turret

The global architecture of the turning-centre is made of two major components:

- the numerical controller manages the interactions with the user and performs all the real-time computations that require some precision
- the PLC controls the operation that requires more complex and flexible computations. In our case, the management of the tool-holder turret is made by a program implemented in the PLC.

The PLC program for this system is represented in fig. 6.

The actions and transition-conditions are described in LD , ST or SFC .

Properties for LD actions and transition-conditions

Some properties can be tailored for the behaviour of the LD language. One of these can be a test for the reachability of each rung of a program. These properties express that there is no dead code in the LD program, plus that the program terminates avoiding infinite loops. The reachability can be expressed the same for every LD program.

Properties for ST actions and transition-conditions

The same remark apply to the ST programs. Reachability properties can be written to avoid dead-code and infinite loops.

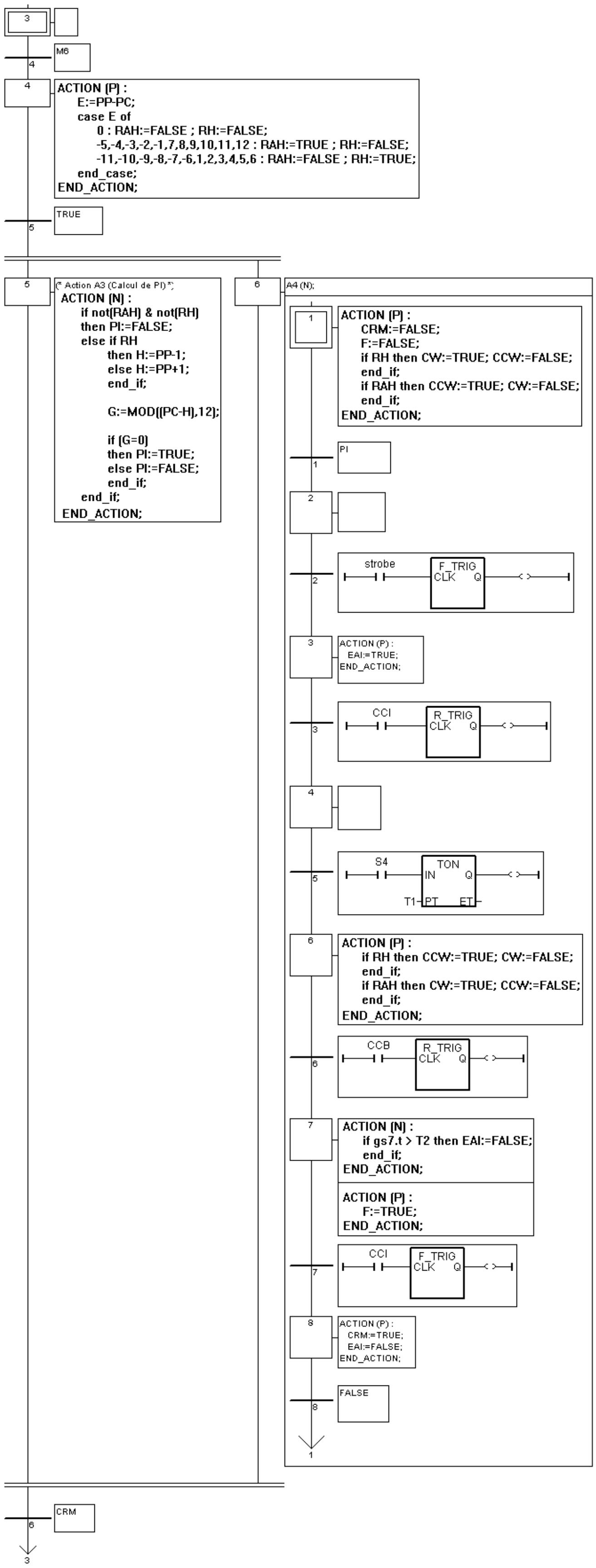

Figure 6: A PLC multi-languages program

Properties for the multi-languages program

Safety properties depend upon the physical system and must be written globally for the complete PLC program based on the requirements.

To verify liveness properties, we have to make assumptions on the behaviour of the inputs variables, thus supposing the corresponding devices work as expected. Some properties can be verified on the example [12]:

- invariant: motor command consistency (to verify that the motor is never ordered in both directions at the same time)
  It is expressed in LTL as:
  $\mathsf{G}\ \neg(CW \wedge CCW)$
- safety: brake-motor consistency (to verify that the motor is always turned off before the brake is on)
  It is expressed in LTL as:
  $\mathsf{G}\ ((eoc \rightarrow \neg Br)\ \mathsf{U}\ (\neg CW \wedge \neg CCW \wedge eoc)) \vee \mathsf{G}\ (eoc \rightarrow \neg Br)$
- liveness: non-blocking system
  It is expressed in LTL as:
  $\mathsf{G}\ ((RH \vee RAH) \rightarrow \mathsf{F}\ CRM)$, plus the fairness assumption on the PI (indexed position) variable:
  $\mathsf{G}\ ((CW \vee CCW) \rightarrow \mathsf{F}\ PI)$

Liveness properties can be made at different levels:

- for each action, we can test the liveness of the sub-program
- for the whole program, we can test that whenever the program start, it will finish

RESULTS

Cadence-SMV checks whether the expected properties are verified on the multi-languages program and if not gives a diagnosis. The result for the invariant property "motor command consistency" is below (made on an average PC workstation):

```
G ¬(CW ∧ CCW).. ......................true

user time......................0.0300432 s
system time....................0.0100144 s

Resources used
==============
user time......................0.0300432 s
system time....................0.0100144 s
```

CONCLUSION

We have developed a formal verification method for PLC programs. Sequential part is described in SFC, actions can be described in the SFC, LD and ST languages and transition-conditions can be described in the LD and ST languages. The principle of our method is the modelling of the behaviour of the whole program as synchronised transition systems: the PLC execution cycle, as well as the languages and theirs elements are modelled. The operational semantics enables the formal and non-ambiguous definition of behaviour, which can then be the base for the coding in temporal symbolic model-checker to verify safety and liveness properties of the program. This method seems to be very efficient to verify PLC programs.

REFERENCES

[1] Bonfatti F., Monari P.D. and Sampieri U., "IEC 1131-3 programming methodology. Software engineering methods for industrial automated systems", *CJ International Editions*, ISBN 2-9511585-0-5, (1997)

[2] Ohman M., Johansson S. and Arzén K.E., "Implementation aspects of the PLC standard IEC 1131-3", *IFAC Control Engineering Practice* 123, Volume 6 n°4, pp. 547-555, (1998)

[3] Babb M., "IEC 1131-3: A Standard Programming Resource for PLC", *Control Engineering*, (1996)

[4] Van Der Wal E., "Structuring Program Development with IEC 1131-3", *PLCopening*, Volume of October, (1998)

[5] Bowen J.P. and Stavridou V., "Safety-Critical Systems, Formal Methods and Standards", *IEE/BCS Software Engineering Journal*, Volume 8 n°4, pp. 189-209 (1993)

[6] Lampérière-Couffin S., Rossi O., Roussel J.-M. and Lesage J.-J., "Formal verification of plc programs: a survey", *ECC'99*, paper N°741, Germany (1999)

[7] Clarke E.M., Wing J.M., "Formal Methods: State of the Art and Future Directions", *ACM Strategic Directions in Computing Research Workshop*, (1996)

[8] Craigen D., Gerhart S. and Ralson T., "An International Survey of Industrial Applications of Formal Methods", Volume 1 "Purpose, Approach, Analysis and Conclusions" & Volume 2 "Case Studies", NIST GCR 93/626 Report, (1993)

[9] Rossi O. and Schnoebelen P., "Formal Modeling of Timed Function Blocks for the Automatic Verification of Ladder Diagram Programs", *4th International Conference Automation of Mixed Processes, ADPM*, (2000)

[10] Lampérière-Couffin S., and Lesage J.-J., "Formal Verification of the Sequential Part of PLC Programs", *5th Workshop on Discrete Events Systems: WODES*, pp. 247-254, (2000)

[11] Rossi O., De Smet O., Couffin S., Lesage J.-J., Papini H. and Guennec H., "Formal verification: a tool to improve the safety of control systems", *Safe Process 2000*, pp. 885-890, (2000)

[12] Canet G., Couffin S., Lesage J.-J., Petit A. and Schnoebelen P., "Toward the Automatic Verification Of PLC programs written in Instruction List", *IEEE International Conference on Systems, Man and Cybernetics*, Nashville USA, (2000)

[13] Bryant R.E. and Musgrave G., "Panel: User Experience with High Level Formal Verification", *35th Design Automation Conference*, pp. 327, (1998)

[14] McMillan K.L., "Symbolic Model Checking", *Kluwer Academic Publishers*, (1993)

[15] Emerson E.A., "Temporal and modal logic", In van Leeuwen J., editor, *Handbook of Theoretical Computer Science*, vol. B, chapter 16, pp. 995-1072, Elsevier Science Publishers, (1990)