LOGIC CONTROLLERS DEPENDABILITY VERIFICATION USING A PLANT MODEL

José J.B. Machado, Bruno Denis, Jean-Jacques Lesage, Jean-Marc Faure, Jaime Fereira

LOGIC CONTROLLERS DEPENDABILITY VERIFICATION USING A PLANT MODEL

José M. Machado [(1)], **Bruno Denis** [(2)], **Jean-Jaques Lesage** [(2)], **Jean-Marc Faure** [(2), (3)], **Jaime C. L. Ferreira Da Silva** [(1)]

[(1)] *University of Minho - Campus de Azurém, 4800-058 Guimarães, Portugal*
[(2)] *LURPA-ENS de Cachan - 61, Avenue du Président Wilson, 94235 Cachan, France*
[(3)] *SUPMECA, 93407 Saint-Ouen, France*

Abstract: This paper focuses on usefulness of a plant model for model-checking of untimed properties of logic controllers. Verification results obtained on a case study by using the symbolic model-checker NuSMV and three methods: verification of the only controller, constraints-based verification, in which the plant is simply modeled as a set of physical constraints, and model-based verification, that relies on a detailed model of the plant, are presented. The results yielded by these approaches enable to draw up application rules for formal verification of logic controllers.

Keywords: Logic controller, formal verification, plant model, model-checking

1. INTRODUCTION

Formal verification of logic controllers thanks to untimed model-checkers has been addressed by numerous researchers (see for instance (Moon, *et al*, 1992), (Bornot, *et al*, 2000), (Lampérière and Lesage, 2000), (Huuck, *et al*, 2003), (Gourcuff, *et al*, 2006)), who produced valuable results, such as formal semantics of the IEC 61131-3 standardized languages (IEC 61131-3, 1993) as well as rules to translate PLC programs into formal models. Most of these works, whose objective is systems dependability improvement, have addressed verification of the only controller and, even when a model of the controlled system, commonly named plant model, has been used in conjunction with the controller (Rausch and Krogh, 1998), (Merkte and Menzel, 2000), no comparison of the results obtained when using and not using this plant model has been performed. The objective of this paper is to contribute to fill this gap.

Adding a plant model to the controller model may a priori leads to fear that combinatory explosion occurs more easily, while on the other hand verification results may be expected more realistic when they are related to the couple controller and plant model. Hence the potential user of model-checking tools may reasonably wonder whether it is useful or not to introduce a plant model when checking properties of a logic controller and, if such is the case, how this model must be constructed and employed. Must it be used for all properties or only for some of them (safety or liveness properties)? Which is the right accuracy level of this plant model?

To contribute to answer these questions, a set of verification experiments has been undertaken. The same set of properties issued from a case study has been verified with the symbolic model-checker NuSMV by using three different approaches. In the first one, only the model of the controller has been verified. In the second one, a set of logical constraints that represents in an abstract manner some significant behaviors of the plant has been introduced. Finally, a detailed model of the plant in the form of a set of state automata has been added to the formal model of the controller. The outcomes that yielded the model-checker as well as the sizes of the state spaces and the verification times has been then compared to estimate the drawbacks and advantages of the two plant models in the frame of formal verification of logic controllers.

The outline of this paper is the following. The case study is briefly presented in section 2. Sections 3, 4 and 5 show respectively how formal models of the

controller, of the properties to check and of the plant can be obtained while section 6 focuses on the translation of these models into the input language of NuSMV. The verification results obtained with the three approaches are discussed in section 7. Conclusions and prospects can be drawn from these results in the last section.

2. CASE STUDY

2.1 Physical system to control

The comparison of the three approaches will be performed thanks to a simple example: an assembly station. The aim of this station is to assembly a gearwheel onto the axle of a mechanical part carried by a pallet that moves on an horizontal conveyor (Figure 1). Its normal automatic operation is the following:

- the incoming pallet is stopped,
- then a manipulator takes the gearwheel and puts it onto the axle,
- the pallet is released while the manipulator moves towards its waiting position (rightmost and uppermost position).

For room reasons, only this normal automatic operation will be considered in what follows.
The manipulator is composed of two cylinders: one horizontal cylinder controlled by an electro-pneumatic bi-stable valve and one vertical cylinder controlled by an electro-pneumatic mono-stable valve. To grip the gearwheel, a vacuum system using suction cups is fastened to the rod of the vertical cylinder. A short cylinder controlled by an electro-pneumatic mono-stable valve permits to stop the pallet at the desired position. When the electro-valve is not actuated, this cylinder is in its uppermost position and the pallet can't move. When this valve is actuated, the cylinder moves down that releases the pallet stopped inside the station or allows any incoming palette to cross freely the station.

2.2 Controller specification

The list of the inputs and outputs of the logic controller is given below while its specification, according to the IEC 60848 standard (IEC 60848, 1988) is presented in figure 2 ("/a" means "not a").

Controller inputs:

pallet_at_assembly_station (p_a_s)	presence_gearwheel (p_g)
vertical_cylinder_down (v_c_d)	vertical_cylinder_up (v_c_u)
horizontal_cylinder_left (h_c_l)	aspiration_on (ason)
horizontal_cylinder_right (h_c_r)	palette_stopped (p_s)

Controller outputs:

RELEASE_PALETTE (R_P)
MOVE_DOWN_VERTICAL_CYLINDER (D_V_C)
MOVE_LEFT_HORIZONTAL_CYLINDER (L_H_C)
MOVE_RIGHT_HORIZONTAL_CYLINDER (R_H_C)
ASPIRATION (ASP)

2.3 Expected properties

In our case, formal verification of the logic controller implies to check whether the both following sets of properties hold or not.

The first set is related to correctness of the IEC 60848 Function Chart of Figure 2 and is the following:

- each step must be reachable (PROP_R_1);
- there is not deadlock (PROP_R_2);
- one and only one step is always active in the steps set (1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14) (PROP_R_3);
- one and only one step is always active in the steps set (5, 6, 7, 8, 9, 10, 12, 13) (PROP_R_4);
- if 11 is active, then 2 must be active too (PROP_R_5);
- if 2 is active, then one of the steps (3, 4, 5, 6, 7, 8, 9, 10, 11) must be active too (PROP_R_6).

Satisfaction of all these properties will ensure correct evolution of the Function Chart.

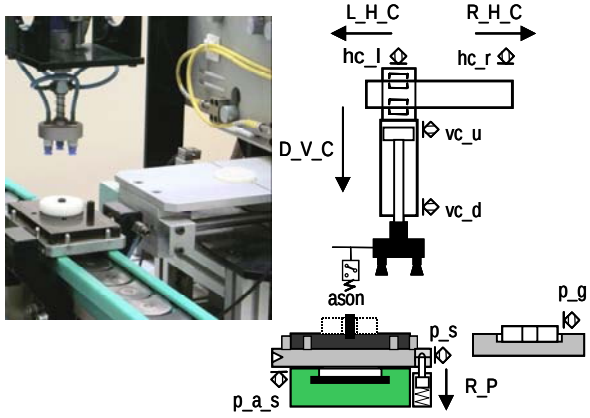

Fig. 1. Controlled plant picture and layout

The second set of properties is related to correct operation of the actuators of the station:

- the two opposite commands of the horizontal cylinder must never be simultaneously set (PROP1);
- when the vertical cylinder moves down, the horizontal cylinder must not move (PROP2);
- when the vertical cylinder is at its lowest position, the horizontal cylinder must not move (PROP3);
- when the horizontal cylinder moves to the right or to the left, the vertical cylinder must always be at its uppermost position (PROP4);
- the manipulator takes the gearwheel at the picking position and doesn't release it until the placing position is reached (PROP5).

The meaning of the first property is obvious. PROP2, PROP3 and PROP4 have been introduced to avoid collisions between the manipulator and its environment that includes several mechanical devices standing between the picking and placing positions. At last, PROP5 describes the correct pick-and-place movement, without dropping the gearwheel.

3. ALGEBRAIC REPRESENTATION OF THE CONTROLLER SPECIFICATION

The assembly station is controlled by a PLC (Programmable Logic Controller) with a cyclic scan monitor. The program that is implemented within this PLC is written using one of the standardized languages (IEC 61131-3, 1993) such as Ladder

Diagram, Instruction List or Structured Text. Whatever the programming language would be, each scan cycle includes three main phases: inputs reading, program execution and outputs updating (Figure 3).

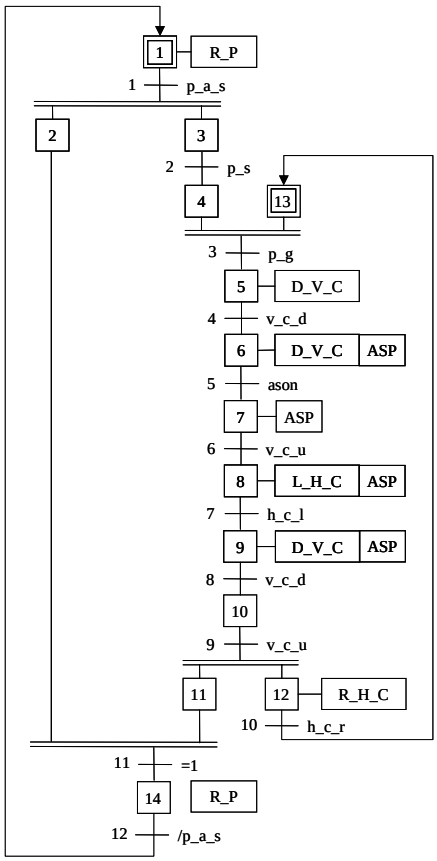

Fig. 2. SFC specification of the controller

As IEC 60848 SFC is a specification language (and not a programming one), it matters to translate the above specification into a program written into a PLC language. This can be done by using the algebraic representation detailed below. In that case, the program will encompass three modules which will be sequentially executed (figure 3): computation of clearing conditions of the transitions, computation of the step variables, and computation of actions.

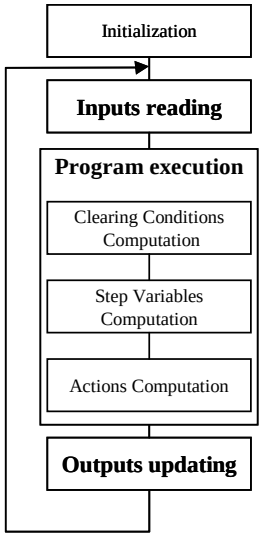

Fig. 3. Cyclic scan monitor of the PLC

3.1. Clearing conditions computation

Let CC(q) (Clearing Condition) a Boolean variable associated to each transition of a SFC. A transition q (Fig. 4) can be cleared if it is enabled (all the steps that precede immediately this transition are active) and if its associated transition condition TC(q) is true. So, in a general case CC(q) can be formulated as follows:

$$CC(q) = (\prod_{j=1}^{m} X_j).TC(q)$$

with:

- X_j: step Boolean variable associated to step j,
- TC(q): Transition Condition associated to the transition q,
- m : number of steps that precede immediately step j.

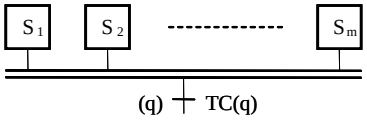

Fig. 4. Transition condition after simultaneous sequences

3.2. Step variables computation

According to the IEC 60848 evolution rules, the Boolean step variable X_i associated to each SFC step i can be computed in the following manner:

$$X_i(t+1) = \sum_{j=1}^{p} CC(p_j) + X_i(t).\prod_{k=1}^{n} \overline{CC(n_k)}$$

with:

- $X_i(t)$: value of the step variable of step i for the t^{th} scan cycle,
- $X_i(t+1)$: value of the step variable of step i for the $(t+1)^{th}$ scan cycle,
- p: number of transitions that precede step i (Fig. 5),
- n: number of transitions that follow step I (Fig. 5),
- $CC(p_j)$: Clearing Condition of the transition (p_j),
- $CC(n_k)$: Clearing Condition of the transition (n_k).

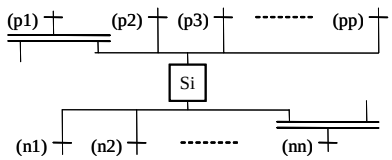

Fig. 5. Step activation/desactivation

In the case of step 5 of the above SFC, for instance, it comes then:

$CC(3) = X_4 . X_{13} . p\_g \qquad CC(4) = X_5 . v\_c\_d$

$X_5(t+1) = CC3 + X_5(t) . /CC(4)$

$X_5(t+1) = (X_4(t). X_{13}(t). p\_g) + (X5(t) . / (X5(t). v\_c\_d))$

3.3. Computation of actions

Each action is set when the logical OR of the step variables of the steps to which this action is associated is true. For instance:

$$D\_V\_C(t) = X_5(t) + X_6(t) + X_9(t)$$
$$ASP(t) = X_6(t) + X_7(t) + X_8(t) + X_9(t)$$

4. PROPERTIES FORMALIZATION

This section is aiming at giving the formal expressions of the two sets of properties presented in natural language in section 2.3. Properties PROP_R_1 to PROP_R_6, that are related to the correct evolution of controller model, can be easily formally translated:

PROP_R_1: $\forall i \in \{1, ..., 14\}, \exists t \in IR^{+*} \mid X_i(t) = 1$

PROP_R_2: $\forall i \in \{1, ..., 14\}, \forall t \in IR^{+*} \mid (X_i(t) = 1), (\exists t' \in IR^{+*}, t'>t, \mid X_i = 0)$

…

PROP_R_5: $\forall t \in IR^{+*}, X_{11}(t) = 1 \Rightarrow X_2(t) = 1$

…

Properties PROP1 to PROP5 deal with the correctness of actuators control. Hence their formal expressions will involve the inputs and outputs of the controller. The formal expressions of properties PROP1 to PROP3 are obtained straightforward given the inputs-outputs list of the controller:

PROP1: $\forall t \in IR^{+*}$, L_H_C . R_H_C = 0

…

Deriving formal expressions for PROP4 and PROP5 requires more care. To build the formal expression of PROP4, an auxiliary variable “Authorization to Move Down”(AMD) is introduced. This variable is a combination of input-output variables as presented below in the Karnaugh table:

L_H_C, R_H_C \ h_c_l, h_c_r	10	11	01	00
00	1	0	1	0
01	0	0	1	0
11	0	0	0	0
10	1	0	0	0

From this table, the expression of AMD is easily obtained:

AMD = (h_c_l . /h_c_r . /R_H_C) + (/h_c_l. h_c_r . /L_H_C)

The formal expression of property PROP4 is therefore:

$\forall t \in IR^{+*}$, D_V_C $\Rightarrow$ AMD ;

$\forall t \in IR^{+*}$,

D_V_C $\Rightarrow$ ((h_c_l . /h_c_r . /R_H_C) + (/h_c_l. h_c_r . /L_H_C))

Property PROP4 is expressed formally by using only combinatory operators. This is not possible for PROP5 that involves a sequence of events. Formalization of this property will be then performed by introducing an auxiliary observer state machine (Figure 6). This state machine describes both correct and faulty behaviors and includes three states. States 100 and 101 and transitions between these states describe the correct operation. When state 102 is active, PROP5 is not verified. The variable P associated to each state shows whether PROP5 is verified or not, P true meaning that the property holds in this state and P false that the property does not hold. The formal expression of PROP5 is given by the state machine of figure 6 and the equation:

$\forall t \in IR^{+*}$, P = 1

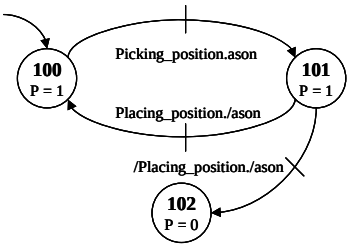

Fig. 6. State machine for the formalization of PROP5

5. PLANT MODEL

As mentioned in (Frey and Litz, 2000), introducing a plant model when checking properties of logic controllers may be carried out according to the two following approaches.

- constraints-based approach in which the plant is modeled by a set of Boolean constraints that describe some relevant physical behaviors, e.g. the two opposite position sensors of a cylinder never deliver simultaneously a true information;
- model-based approach, in which the plant behavior is modeled in a more detailed fashion, for instance in the form of state automata.

Both approaches will be employed in what follows. Moreover, to facilitate plant model construction, a modular method has been developed. In this method a module may be, for instance, a cylinder with its two position sensors and its command electro-valve (Figure 7). Reader could get detailed information about this method in (Machado *et al.*, 2006).

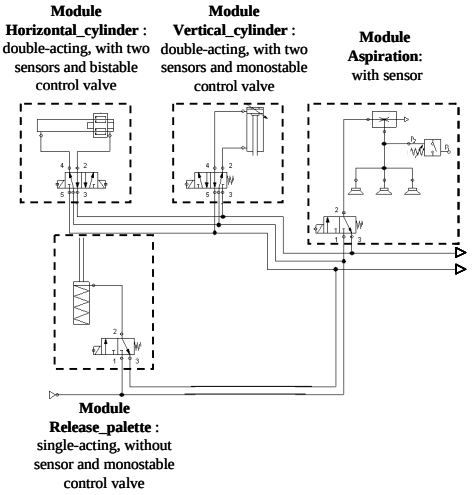

Fig. 7. Modular plant decomposition

5.1 Constraints-based approach

In this case, the models of the horizontal (vertical) movement points out merely that the information provided by the two position sensors at the leftmost and rightmost (top and bottom) positions are never simultaneously true:

$\forall t \in IR^{+*}$, (h_c_l . h_c_r = 0)

$\forall t \in IR^{+*}$, (v_c_u . v_c_d = 0)

No other constraints have been introduced.

5.2 Model-based approach

Each plant module is then modeled as an automaton that describes its different physical states and the transitions between these states The model of the horizontal movement for instance (figure 8) includes four states that correspond to the rightmost and leftmost positions as well as to the movements from one of this position to the opposite one.

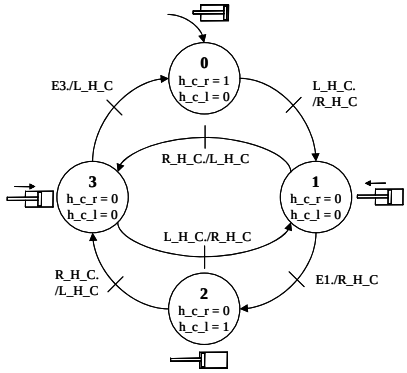

Fig. 8. Horizontal movement model

In this model, variables E1 and E3 are introduced so as to design a stable model (Machado, *et al*, 2006).

6. DESIGN OF THE NuSMV CODE

The formal models of the controller, of the plant and of the properties can be easily translated into NuSMV language (Cimatti, *et al.*, 2002).

6.1 Controller coding

The NuSMV model of the controller is issued from the algebraic equations presented in section 3.

Variables declaration: The controller input variables and the step variables are declared as Boolean.

```
MODULE controller (p_a_s, p_s, p_g, v_c_d, v_c_u,
h_c_l, h_c_r, ason)
VAR
        p_a_s : boolean;
        ...
        X1 : boolean;
        ...
```

Step variables treatment: These variables are initialized according to the SFC model. Each algebraic evolution equation gives rise to one NuSMV statement.

```
        ASSIGN
        init(X1) := 1;
        ...
        init(X14) := 0;
        next(X1) := CC12 | (X1 & !CC1);
        ...
```

Definitions: The values of the outputs and of the auxiliary variables are computed from those of the inputs and of the steps variables.

```
DEFINE
        CC1 := X1 & p_a_s;
        ...
        CC12 := X14 & !p_a_s;
        D_V_C := X5 | X6 | X9;
        ...
```

6.2 Plant model coding

Logical constraints translation is straightforward, if the constraints-based approach is selected. When the plant model is represented by a set of automata, each of them can be translated as described below, for the horizontal movement.

```
MODULE Horizontal_cylinder_MB (L_H_C, R_H_C)
VAR
   L_H_C: boolean;
   R_H_C: boolean;
   E1 : boolean;
   E3 : boolean;
   State : {left, moving_right, right, moving_left};
ASSIGN
init(state) := right;
next(state) := case
state = right & L_H_C & ! R_H_C: moving_left;
state = moving_left & E1 & ! R_H_C : left;
state = moving_left & R_H_C & ! L_H_C: moving_right;
state = left & R_H_C & ! L_H_C: moving_right;
state = moving_right & E3 : right;
state = moving_right & L_H_C & ! R_H_C: moving_left;
1 : state;
   esac;
DEFINE
   h_c_l := state=left;
   h_c_r := state=right;
```

6.3 Properties coding

The properties can be formalized by temporal logic expressions (Clarke *et al.*, 1986) in Computation Tree Logic (CTL) or Linear Temporal Logic (LTL) (Emerson and Halpern, 1986). Both are well defined in (Bérard *et al.* 1999)):

```
PROP_R_1: EF Xi
PROP_R_2: AG (Xi ⇒ EF ¬Xi)
...
PROP_R_5: AG X11 ⇒ X2)
...
PROP1: AG ¬ (R_H_C ∧ L_H_C)
...
PROP4: AG D_V_C ⇒ ((h_c_l ∧ ¬h_c_r ∧ ¬ R_H_C) ∨
(¬h_c_l ∧ h_c_r ∧ ¬L_H_C))
PROP5: AG P
```

These CTL statements can be directly coded into NuSMV language as follows:

```
SPEC EF X1; -- PROP_R_1_X1
...
SPEC EF X14; -- PROP_R_1_X14
SPEC AG (X1 -> EF !X1); -- PROP_R_2_X1
...
SPEC AG (X1 -> EF !X14); -- PROP_R_2_X14
SPEC AG (X11 -> X2); -- PROP_R_5
...
SPEC AG !(R_H_C & L_H_C); -- PROP1
...
SPEC AG  D_V_C ->((h_c_l & ! h_c_r & ! R_H_C)
|(!h_c_l  &  h_c_r  & ! L_H_C)); -- PROP4
SPEC AG P --  PROP5
```

7. VERIFICATION RESULTS

Once the formal models of the controller, of the properties and of the plant designed, it is possible to check whether the two sets of properties hold or not:

- on the model of the only controller (approach non-model-based);
- on the model of the whole system {controller + plant} when some parts of the plant are modeled as a set of constraints (approach constrained-based);

– on the model of the whole system {controller + plant} when the plant is modeled in a detailed fashion (approach model-based).

The results that yield these three approaches are given in Table 1.

Table 1 – Results for the three verification approaches

Properties	Non model-based	Constraints-based	Model-based
PROP_R_1	true	true	**true**
PROP_R_2	true	true	**true**
PROP_R_3	true	true	**true**
PROP_R_4	true	true	**true**
PROP_R_5	true	true	**true**
PROP_R_6	true	true	**true**
PROP1	true	true	**true**
PROP2	true	true	**true**
PROP3	**false**	**false**	**true**
PROP4	**false**	**false**	**true**
PROP5	true	true	**true**
Reachable states	19456	10994	23552
Computing time (s)	2,7	2,4	23,4

Several significant conclusions can be drawn up from these experiments. First, as pointed out clearly in the table, constraints based approach gives the same results than non-model-based approach but both state space size and computing time are shortened with the first approach. Hence introducing constraints which roughly model the plant can reduce memory needs and verification time.

Two safety properties (PROP3 and PROP4) are only verified when the model-based approach is employed. This does not mean that the controller specification includes design errors but merely that it has been designed assuming that the controller is coupled with a plant which generates relevant signals for each SFC situation, e.g. a true level of a position sensor at the end of the movement leading to this sensor. This enables to state that, generally speaking:

– a negative proof with non model-based verification means either that the controller includes design errors or that is correct, but assumed to be connected to a non-faulty plant;
– a positive proof with non model-based verification is more meaningful than with model-based verification. In the first case indeed the property holds whatever the plant behavior would be (correct or faulty); in the second case, the property is only verified when the plant behaves in the right manner.

At last, it shall be noted that model-based verification does not increase significantly the state space size but slows down verification.

8 CONCLUSIONS AND PERSPECTIVES

In this paper we showed that the use of a plant model has a great impact on formal verification of discrete event systems. In scientific literature, model-based and non model-based approaches are often brought into conflict but never compared. We pointed out through a study case that in fact those two approaches complement each other. Furthermore, we showed which approach must be preferred depending on class of properties.

Nevertheless, model-based verification asks the problem of the construction of the plant model. Current and future works aim at producing generic models and modular method for the design of such plant model: including time or not, deterministic or not, including faulty behavior or not.

REFERENCES

B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. (1999). Systems and software verification: model-Checking techniques and tools. Springer, 1999.

S. Bornot, R. Huuck, Y. Lakhnech, B. Lukoschus. (2000) Verification of sequential function charts using SMV, *Proc of PDPTA'2000*, 2000.

A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. (2002). NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. *Proc. CAV'02*, Copenhagen (Denmark), July 2002.

E.M. Clarcke, E.A. Emerson, A.P. Sistla. (1986). Automatic verification of finite-state concurrent systems using temporal logic specification. *ACM Transaction on Programmation Languages and System*s, **Vol. 8**, n°2, pp. 244-266, 1986.

E.A Emerson and J.Y. Halpern. (1986). Sometimes and Not Never revisited : on branching versus linear time temporal logic. *Journal of the ACM*, **33**, 1, p. 151-178.

G. Frey and L. Litz. (2000). Formal method in PLC programming. *Proc. of IEEE SMC'2000*, CDRom paper n°2431, 6 pages, October 8-11, Nashville, Tennessee-USA.

V. Gourcuff, O. de Smet, J.-M. Faure (2006). Efficient representation for formal verification of PLC programs. *Proc. of WODES 2006*, July 10-12, Ann Arbor, USA.

R. Huuck, B. Lukoschus, and N. Bauer. (2003). A model-checking approach to safe SFCs. *Proc. of CESA'03*, Lille (France), July 2003.

IEC 60848. (1988). Preparation of function charts for control systems. IEC Standard, 1988.

IEC 61131-3. (1993). Programmable Controllers – Programming languages. IEC Standard, march 1993.

S. Lampérière, J.-J. Lesage. (2000). Formal verification of the sequential part of PLC programs. *Proc. of 5th IFAC Wodes*, pp. 247-254, Ghent (Belgium), August 2000.

J. Machado, B. Denis and J.-J. Lesage (2006). A generic approach to build plant models for DES verification purposes. *Proc. of WODES 2006*, July 10-12, Ann Arbor, USA.

I. Moon, G.J. Powers, J.R. Burch, E.M. Clarke. (1992). Automatic Verification of Sequential Control Systems Using Temporal Logic, *AIChE Journal*, **Vol. 38**, n°1, pp.67-75, 1992.

M. Rausch, B. H. Krogh. (1998). Formal Verification of PLC Programs, *Proc. of AAC'98*, Philadelphia, PA, USA, June 1998.

T. Merkte and T. Menzel. (2000). Methods and tools to the verification of safety-related control software. *Proc. IEEE SMC'00*, pp. 2455-2457, Nashville, Tennessee, USA, October 2000.