Towards IEC 61499 Function Blocks Diagrams Verification

Camille Schnakenbourg, Jean-Marc Faure, Jean-Jacques Lesage

Towards IEC 61499 Function Blocks Diagrams Verification

C. Schnakenbourg[*,***], J.-M. Faure[**,***], J.-J. Lesage[***]

[*]CNAM 21 rue Pinel F75013 Paris, France

[**]ISMCM-CESTI, 3 rue Fernand Hainaut, F93407 Saint-Ouen Cedex, France

[***]LURPA, ENS de Cachan, 61 avenue du Président Wilson, F94235 Cachan Cedex, France

[schnakenbourg, faure, lesage]@lurpa.ens-cachan.fr

Abstract -- **After having sketched the different techniques enabling to check properties of Discrete Event Systems control software, we present in this article a formal method for IEC 61499 function blocks diagrams verification. This method is based on a formal representation of the behaviour of function blocks diagrams and takes benefit of verification tools developed from the SIGNAL synchronous language.**

Keywords: **Properties Verification, Function Blocks Diagram, IEC 61499 Standard, Synchronous Languages, Control Systems Safety**

I. INTRODUCTION

Since numerous years safety is a major industrial issue. If it is true that industries like transport, critical industries (energy or chemical processes for instance) must, more than others, be concerned by safety objectives, in fact all the industries must consider this subject. So, in order to handle safety, each industrial field has developed some particular standards. The growing demand for automated systems has led to take into account control systems safety too. Nowadays industry needs not only safe processes but also safe control systems.

To federate safe control systems development, the recent IEC 61508 standard [1] proposes a generic model that can be applied to all the Electrical /Electronic/ Programmable Electronic (E/E/PE) safety-related systems. This standard provides a generic framework within which accurate methods can be applied, whatever the application domain should be, and shows moreover that safety must be an everyday preoccupation during all the life of a system.

Considering the great amount of methods developed in order to design and implement safe control systems, classification criteria have to be defined to organise these methods. We will focus only on two classifications based on complementary points of view: the control system life cycle and the expected objectives.

With a life cycle point of view, [2] introduces the On-line and Off-line safety notions based on the classical "square root" life cycle (Figure 1). This cycle is composed of two different phases: the conception phase and the exploitation phase. To these two phases are linked two kinds of safety: Off-line safety for the first phase, On-line safety for the second one.

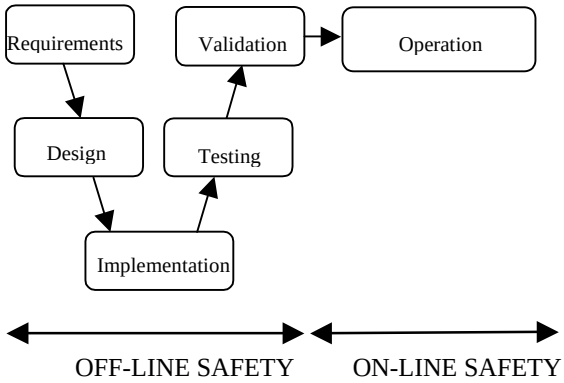

Figure 1 ***Life cycle and safety [2]***

The purpose of the *Off-line safety* methods is to minimise the fault risk during conception, i.e. before the system is used. On the opposite side, the objective of the *On-line safety* methods is to ensure safety in an already implemented and running system. This article deals only with *Off-line safety* methods.

On another hand, Laprie [3] proposes a classification based on the expected objective of the method. The safety-related works are arranged in four categories: fault prevention methods, fault forecasting methods, fault tolerance methods and fault removal methods.

Fault prevention methods aim at organising development process and are rather related to system engineering. Functional analysis, project organisation and management methods, business process engineering are examples of such methods.

Fault forecasting methods can be split in two categories: ordinal assessment methods, like FMECA, and probabilistic assessment methods, that provide numerical values of safety attributes (reliability, availability). These last ones use models such as fault trees, Markov chains, stochastic Petri nets.

Fault tolerance methods are based on the assumption that the occurrence of a failure shall not stop the system operation. These methods fit well with systems owning some flexibility degrees and/or redundancies.

Fault removal when designing and implementing a system consists in verifying the results of the different activities (models, drawings, schemata, programs, sub-systems), by tests, simulation techniques or formal verification methods. During the

operation of the system, fault removal is performed by corrective and preventive maintenance.

This article will focus on the off-line fault removal methods for control software. In the first part, simulation and verification methods are shortly described. This enables to point out the advantages and the drawbacks of these two kinds of methods and leads to strongly advocate verification for safety-related systems. The second part is dedicated to the presentation of a formal method developed in our laboratory in order to verify programs developed in the IEC 61499 function blocks language.

II. CONTROL SOFTWARE SIMULATION AND VERIFICATION

Industrial control programs are developed by using standard languages described in the IEC 61131-3 or IEC 611499 [4] standards in order to be adapted to the industrial needs. Whatever the chosen language could be (Sequential Function Chart, Ladder Diagram, or another standard language), a control program behaves as a Discrete Event System and hence can be represented by a state automaton. It is the reason why software simulation and verification methods will be explained thanks to this suitable representation of the behaviour of the control software.

A. Simulation

This method consists in stimulating the program by inputs sequences representing the behaviour of the controlled process and in checking whether the outputs sequences generated by the program are compliant to the application requirements or not. Simulation techniques are very popular in industry. They take benefit of specialised software providing process simulation with user-friendly interfaces. These software tools facilitate the generation of input sequences and the interpretation of the results. Nevertheless the main drawback of simulation is not to enable a complete verification of the program behaviour within a reasonable time. This comes from the huge amount of input sequences that should be generated to test the entire program.

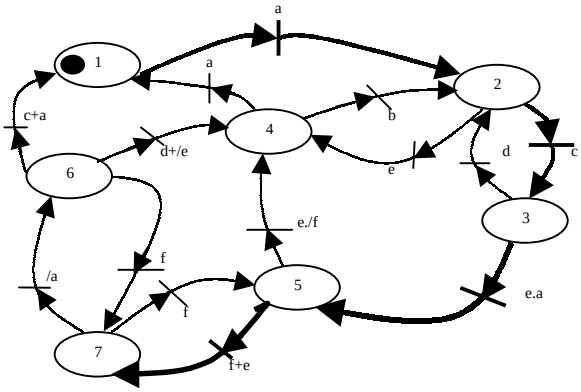

Figure 2 ***Simulation drawback***

A simple explanation can be given thanks to the state automaton depicted in Figure 2 and assumed to represent part of the equivalent automaton of a control software. In this state space, simulation is aimed at generating a set of paths and at verifying if properties are satisfied along these paths.

Unfortunately the huge size of state automata representing industrial control software does not enable to go along all possible paths during a simulation session. In the example of the state automaton depicted in Figure 2, simulation allows to check properties related to the path in bold but does not give any information on properties related to the other paths.

So, simulation is not an exhaustive method and the quality of a simulation session is up to the skill and the experience of the automation engineer who has to choose relevant inputs sequences corresponding to usual and hazardous situations of the controlled process. Formal verification methods have been developed to tackle this problem by providing means allowing to verify the totality of a program.

B. Formal verification methods

Three kinds of formal verification methods are usually defined :

- theorem-proving or algebraic methods
- model-checking
- methods based on the translation of the model or the software to be verified into a formal language, e.g. a Petri Net or a synchronous language.

The last one is merely aimed to take benefit of formal analysis tools developed from the target language. Hence only theorem-proving and model-checking are really basic verification methods that enable an exhaustive analysis.

Whatever the verification method could be, formal expression of the software behaviour and of the properties is required. Properties can be grouped into three categories [5]: vivacity and safety (something must or must not happen) and celerity (time related properties). *A priori*, those properties are defined in the requirements. All the verification methods need that the properties are written in a formal way. With this goal, [6] introduces a formalism to express dynamic properties: the temporal logic. This logic enables to write formulas describing in a formal way expressions including words like “until”, “always”. CTL* [7] is an example of such a logic.

The algebraic methods’goal (a.k.a. theorem-proving) is to do or to verify proofs, manipulating only the syntax as it can be done in a mathematics’ demonstration. A prover takes a hypothesis (H) as data or definitions and a formula or a property to be proved ϕ and search if ϕ can be obtained from H using the deduction rules of the used logic. One of the interests of such a method is that no hypothesis on the model to analyse is to be made. More particularly infinite state automata can be proceeded. The main drawback is indecidability, which means that no solution is found in some cases. An example of theorem-proving can be found in [8].

Model-checking operates on state automata. The basis principle of this method is the marking process explained hereafter.

On the considered automata, for a property ϕ, composed of sub-properties ψ_i, each state q verifying ψ_i will be marked thanks to a variable $q.\psi_i$, true when ψ_i is verified on q. Thus, $q.\phi$, which indicates whether the property is verified or not, can be obtained from the $q.\psi_i$. More precisions on model checkers for the temporal logic CTL can be found in [9].

All the model-checking algorithms, and more generally all the algorithms using explicitly given state automata share the same problem: they hugely depend on the size of the automata. Thus, the marking time is a function of the number of states and transitions of the automata. Even if it is possible to reduce a state automaton, the combinatory explosion makes classical model-checking methods unusable for the verification of huge industrial systems.

In order to overcome this problem, a new kind of model-checker named symbolic model-checker has been created. Symbolic model-checkers use a symbolic representation of the automata (for example equations or BDD [10] [11]). This kind of formal method can handle the equivalent of a 10^{22} state automata.

Another drawback of model-checking methods is that the languages used by model-checkers are not industrial languages (there is no IEC 61131-3 model-checker for example).

So, the last solution consists in the translation of the model or software to verify into a language from which formal verification tools (theorem-prover or model-checker) have been developed. Synchronous languages offer this possibility.

Our laboratory has several results for each verification method: theorem-proving and model-checking (for example [12], [13], [14]). In the following, an example of properties verification on an industrial standard language using the synchronous language SIGNAL will be presented.

III. IEC 61499 FUNCTION BLOCKS DIAGRAMS VERIFICATION

This work is an abstract of the work presented in [14]. Its goal is to verify properties on IEC 61499 function blocks models. This objective can only be achieved through a formally defined syntax and semantics for this function block language.

As proposed in [15], this formalisation problem can be solved thanks to a new class of Petri Net named Signal/Net Systems. The main drawback of such a method is that the verification tools don't exist and must be developed too. So we decided to use results on synchronous languages. More precisely, some similarities between IEC 61499 function blocks diagrams and SIGNAL process diagrams having been remarked, the development of a verification method based on the synchronous language SIGNAL [16] and on the (Z/3Z, +, *) algebra [17] was undertaken.

This method consists mainly in (Figure 3) :

- translating the function blocks diagram into a SIGNAL model ;
- expressing the properties that must be verified into a SIGNAL syntax ;
- exploiting the existing proof tools that use the automaton equivalent to the SIGNAL model.

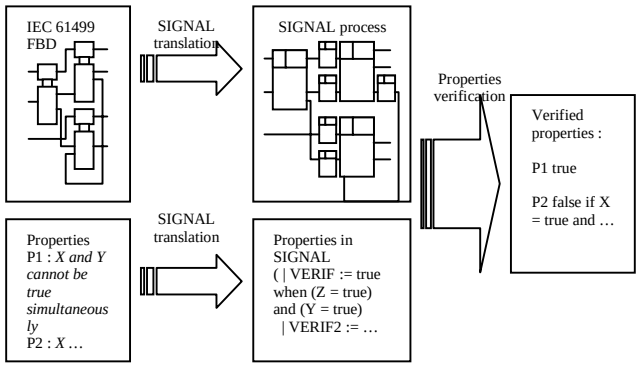

Figure 3 ***Verification method***

Such a method has already been used to prove properties for models described in Statecharts or in GRAFCET [18]. Though the final objective would be formal verification, the translation of function blocks diagrams into SIGNAL processes allows, in a first step, the simulation of their behaviour thanks to SIGNAL models simulators, like SILDEX© from the TNI society. This feature can enable a better understanding of negative verification results.

A. A brief description of the IEC 61499 standard [4]

The IEC 61499 standard draft focuses on industrial-process measurement and control systems and proposes, to that purpose, several concepts including the function block[1] (Figure 4 a)).

A function block is built using two parts: the ECC ("Execution Control Chart") part shown in Figure 4 b) which receives and sends events, and an algorithmic part which receives and sends data. The ECC, which is a state automaton included in the ECC part, is described in an IEC 61131-3 SFC like language [19], for the construction rules (excluding parallelism) as for the behaviour rules. This behaviour model, composed by a succession of EC-states and EC-transitions, can control the algorithm part using the input events (the variables Ev1 to Ev4 in our figure, named Event Input variables or EI-variables). The ECC can, eventually, send some output events using event output variables, or EO-Variables (EvO2 in the example) or modify some internal variables. Those internal variables as the output events can be used in the receptivities associated with the ECC transitions. The algorithm part describes the relations between the input data (Da and Db in our example) and the output data (Dc in the example). Those algorithms are expressed using the IEC 61131-3 languages (IL, SFC, Ladder diagram…). In the example, the two algorithms A1 (Dc = Da + Db) and A2 (Dc = Da – Db) are used when the EC-states E1 and E2 are activated respectively. An output event is sent when E2 is activated.

[1] The IEC 61499 function blocks must be distinguished from the IEC 61131-3 function blocks. The first one is a structure entity using a state automaton to describe only a part of its behaviour. For the second standard, a function blocks diagram can be linked to a step of a SFC.

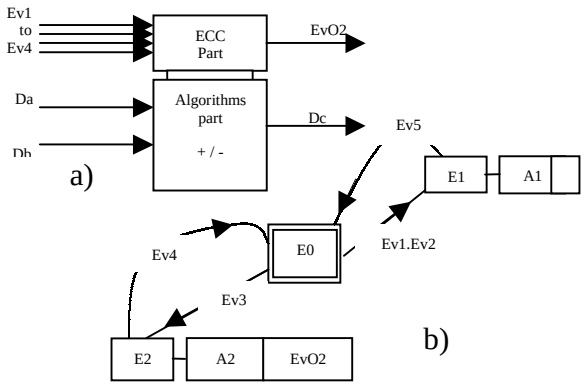

Figure 4 ***a) block example b) ECC of this function block***[2]

The blocks can be linked to build nets named function blocks diagrams (Figure 5). The main building rules are:

- The block outputs are linked to the inputs of another block.
- An event type output can only be linked to an input of the same type (this rules also applies to the data).
- An input (event or data) can only be linked to one and only one output of the same type.

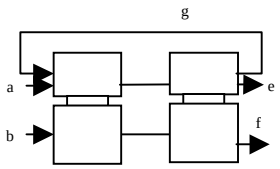

Figure 5 ***A simple function blocks diagram***

The syntactic formalisation of those rules can be made thanks to a static metamodel. This metamodel will not be presented in this article.

The standard introduces several other concepts (scheduling function, communication blocks, …) whose aim is to implement the diagram in a distributed real time system. Those concepts, mandatory when implementing a control system, will not be used in this paper, for only control design will be considered.

B. Function Blocks Diagram translation

Figure 6 depicts the chosen methodology. The analysis of the standard enables to define two evolution algorithms: one for the function block (FB) and one for the function blocks diagram (FBD). These algorithms explain formally the generic behaviour of a block or a diagram. Then it is possible to translate any block or block diagram into a SIGNAL model thanks to these formal behaviours definitions.

[2] In the case where no output event is associated to a state, an empty rectangle is left where normally the name of an output event should take place (see the E1 EC-state).

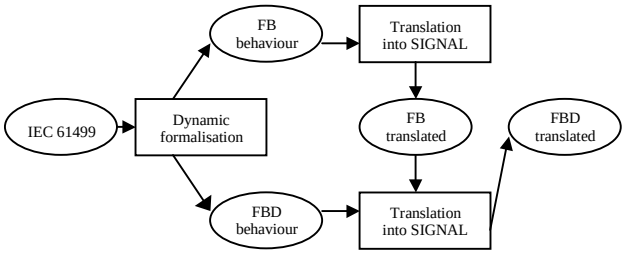

Figure 6 ***Translation method***

Figure 7 sketches the function block evolution algorithm that includes two loops. The first one models input events reading, the second one the ECC evolution. It matters to highlight that several evolutions of this last loop (numbered 2 in figure 7) can happen before two successive inputs reading. This feature, defined in the standard in order to ensure a **deterministic behaviour**, means merely that an ECC stable state must be reached before reading the inputs. So this FB evolution algorithm can be called "Stability-driven".

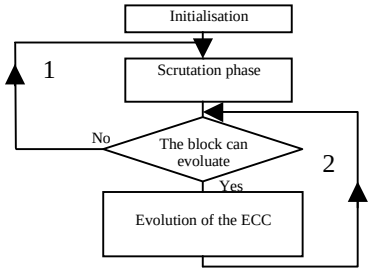

Figure 7 ***Behaviour of a block***

From this algorithmic representation, a global translation method from a FB into its SIGNAL equivalent model has been developed. The main phase of the algorithm is the ECC evolution phase, expressed below in a SIGNAL syntax:

Xi = true when (((Xi$=true) and (Ri$=false)) or
((X(i-1)$=true) and (R(i-1)$=true))) default false

Where Xi is a state variable true when the EC-state is active, Ri is the receptivity of the downstream transition of the state i. This equation is nothing else than the translation into SIGNAL of the behaviour of a finite state automaton evolving according to IEC 61499 rules.

However, these signals have only a sense if they are linked to a given clock. That's why a clock, named Block Clock (Cbl), synchronised with the loop number 2 and with which are associated all the variables of the here-above equation, is to be defined. However this clock addresses only internal variables of FB. To translate an entire FBD into SIGNAL, an other clock related to FB input and output events must be defined. The definition of this clock as well as the relation between block clock and event clock is presented below.

Event and block clocks

No assumption is made on FB input events clocks: they can be different. In order to simplify the SIGNAL model, a clock named event union clock and noted Cunion, equal to the union of the different clocks of the input events, has been defined. Figure 8 depicts the construction of Cunion.

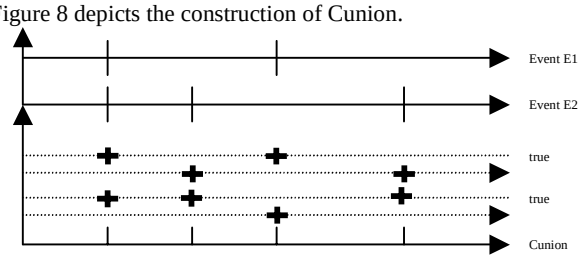

Figure 8 ***Cunion clock definition***

To synchronise an input event with this clock is equivalent to create a Boolean signal, whose clock is Cunion, true when the event is present, false otherwise.

The IEC standard indicates also that any occurrence of an output event brought about by an occurrence of an input event must be sent before the next occurrence of an input event. This imposes to overclock Cunion. For that purpose, a clock named event clock and noted Ceve has been defined from Cunion. That clock has a period equal to a fraction of the lowest interval between two dates of Cunion. Figure 9 shows an instance of Ceve built from the Cunion clock of figure 8 with a period twice smaller than the lowest time interval of Cunion.

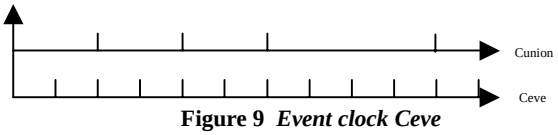

Figure 9 ***Event clock Ceve***

At last, the standard indicates that:

- all the occurrences of input events are to be taken into account (no occurrence is missed), and that
- all the evolutions of a FB coming from a given input event occurrence should be ended before an other input event occurs.

This **reactivity property** leads to consider that the block clock is faster than the event clock so that no input event occurs during a FB evolution (loop number 2 in figure 7).

More precisely, the ratio between the event clock period and the block clock period shall be greater than or equal to the maximum number of evolutions of the block coming from any input event occurrence. This number can be easily determined by analysing the evolutions of the ECC with regards to the algorithm of figure 7. Such a method has already been used in [18].

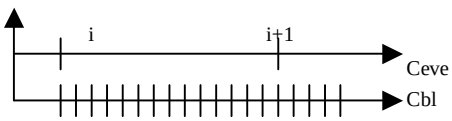

Figure 10 ***Block and event clocks***

Function blocks diagrams modelling

The function blocks diagram evolution algorithm is based on several assumptions:

- All the blocks can evolve in parallel.
- There is no priority of a block on another.
- There is no priority of an algorithm on another.

With these assumptions, it is possible to derive from a FBD a SIGNAL model including as much processes as FB in the initial FBD. Each process encloses a SIGNAL code describing the behaviour of the FB from which it is derived. Each process receives the same input signals and emits the same output signals as the corresponding FB. All the events share the same clock: Ceve, built as explained previously. The block clock is common to all the blocks. As initially the FB can have different clocks (a FB clock can be twice faster than Ceve whereas another one is four times faster), the faster clock is chosen, in order to ensure reactivity for all the FB. Hence, only two clocks are used within the model.

In order to ensure clocks consistency, two processes must been added for the inputs and for the outputs of each process. The aim of these processes is to convert signals with the event clock to signals with the block clock (for the inputs) and vice versa (for the outputs).

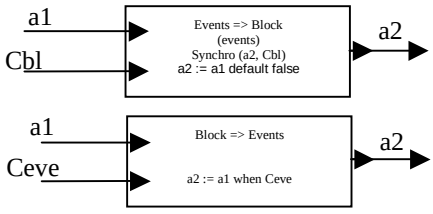

Figure 11 ***Clocks consistency SIGNAL processes***

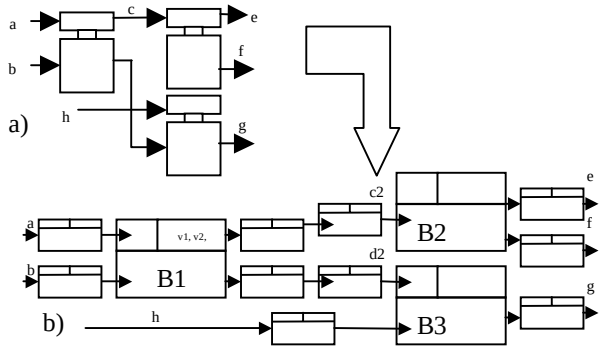

Figure 12 ***a) Function block diagram***

b) SIGNAL process diagram[3]

[3] For clarity reason, the inputs Ceve and Cbl are not shown in the input of the process E->B and B->E.

C. Example

In order to illustrate the developed method, a simplified version of the steam boiler control [20] has been treated. This example has been chosen because:

- It deals with numeric and boolean variables.
- It is close enough to an industrial problem.
- It is related to a critical system.
- It is a reference in the hybrid system control field.

After translation of the specification into an IEC 61499 function blocks diagram, the translation of this diagram into a SIGNAL model has been made. This model includes five blocks, each of them embedding approximately 100 lines of SIGNAL code. Properties have been formally verified on this model by using the SILDEX tool. Each property verification is performed in one to five minutes. To illustrate the usefulness of the approach, some verification results of the property "the failure of all the pumps starts the emergency stop mode" will be given. This property is a vivacity property that had been chosen because it is a major concern for the designers of a nuclear power plant.

With our initial model, the SILDEX tool gives two kinds of results:

- There is a situation where, when all the pumps are out of order, the emergency stop mode is not reached. Hence the studied property is not verified with this model.
- A trace allowing to find how this situation can be reached. This trace is composed of a succession of input events leading to the faulty situation.

These results enable to correct the initial model so that the property would be satisfied for any input events sequence.

For the moment, no property dealing with physical time has been verified because SILDEX considers only logical time. This problem can be overcome by giving a value to the gap between two instants of a clock; e.g. by translating a property such as "the pump must make less that 5 minutes to become operational" into "the pump must take less than 100 periods of Ceve to become operational".

IV. CONCLUSIONS

This article was aimed to highlight the interest of formal verification methods (theorem proving, model checking) for safe control systems development.

The verification method of IEC 61499 function blocks has enabled to show the advantages and drawbacks of the SIGNAL based approach.

Future works will focus on verification of distributed systems and of real time multitasking systems.

V. REFERENCES

[1] International Electronical Committee, "IEC 61508 (1998-2000), Functional safety of electrical/electronic/programmable electronic safety-related systems. Parts 1-7." 1998.

[2] J.-M. Faure, J.-J. Lesage, "Methods for safe control systems design and implementation", IFAC conference INCOM 2001, Vienna (Austria), CD-ROM support, 2001.

[3] Laprie, J.C. and al., "Guide de la surete de fonctionnement", Cepadues, 1996.

[4] International Electronical Committee, "IEC 61499 function blocks for industrial-process measurement and control systems", draft version ; 1999.

[5] L. Lamport, "Proving the correctness of multiprocess programs", IEEE Transactions on software engineering, N° 3(2), pages 125-143 ; 1973.

[6] A. Pnueli, "The temporal logic of programs", Proceedings of the 18th IEEE symposium foundations of computer science (FOCS'77), pages 46-57, Providence, RI, USA; 1977.

[7] E. A. Emerson, J. Y. Halpern," "Sometimes" and "not never" revisited: On branching versus linear time temporal logic", Journal of ACM, N° 33(1), pages 151-178 ; 1986.

[8] X. Rival, J. Goubault-Larrecq, "Experiments with finite tree automata in Coq", Proceedings of the 14th Int. Conf. Theorem Proving in Higher Order Logics (TPHOL'01), pages 362-377, Edinburgh, Scotland ; 2001.

[9] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen, "Systems and Software Verification. Model-Checking Techniques and Tools", Springer, 2001.

[10] O. Coudert, C. Berthet, J.C. Madre, "Verification of synchronous sequential machines based on symbolic execution", Lecture notes in computer science, vol. 407, pages 365-373 ; 1980.

[11] J-R. Bruch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang, "Symbolic model-checking, 10^{20} states and beyond", Information and Computation, N° 98(2), pages 142-170 ; 1992.

[12] S. Lampérière-Couffin, O. Rossi, J.-M. Roussel, J.-J. Lesage "Formal validation of PLC programs: a survey", ECC'99, 1999.

[13] C. Thierry, J.-M. Roussel, J.-J. Lesage, "An extended boolean algebra for the control of logical systems", IMACS'00, 2000.

[14] C. Schnakenbourg, "Formalisation en vue de la vérification de propriétés du langage à blocs fonctionnels IEC 61499", Ph. D thesis, ENS Cachan, 2002.

[15] V. Vyatkin, H.-M. Hanisch, P. Starke, S. Roch, "Formalisms for verification of discrete control applications on example of IEC1499 function blocks", Proceedings of the conference « Verteilte Automatisierung » (Advanced Automation), Magdeburg, Allemagne, pages 72-79 ; 2000.

[16] P. Le Guernic, M. Le Borgne, T. Gautier, C. Le Maire, "Programming real time application with SIGNAL", Proc. of the IEEE, 79(9) :1321-1336, 1991.

[17] M. Le Borgne, "Systèmes dynamiques sur des corps finis", Thèse de doctorat, Université de Rennes I, 1993.

[18] P. Le Parc, L. Marce, "Synchronous definition of GRAFCET with SIGNAL", Proc. of the IEEE/SNC'93 conference, le Touquet, France, 1993.

[19] International Electronical Committee, "IEC 61131-3 standard for programmable controllers. Part 3: Programming language", 1993.

[20] J.-R. Abrial, "Steam-boiler control specification problem", Dagstuhl Meeting: Methods for semantics and specification, 1995.