HAL open science

Efficient representation for formal verification of PLC programs

Vincent Gourcuff, Olivier de Smet, Jean-Marc Faure

To cite this version:

Vincent Gourcuff, Olivier de Smet, Jean-Marc Faure. Efficient representation for formal verification of PLC programs. 8th International Workshop On Discrete Event Systems (WODES'06), Jul 2006, Ann Arbor, United States. pp. 182-187. hal-00175431

HAL Id: hal-00175431

https://hal.science/hal-00175431

Submitted on 28 Sep 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Efficient representation for formal verification of PLC programs *

Vincent Gourcuff, Olivier De Smet and Jean-Marc Faure
LURPA – ENS de Cachan,
61 avenue du Prés. Wilson, F-94235 Cachan Cedex, France
Email: {gourcuff, de_smet, faure}@lurpa.ens-cachan.fr

***Abstract*— This paper addresses scalability of model-checking using the NuSMV model-checker. To avoid or at least limit combinatory explosion, an efficient representation of PLC programs is proposed. This representation includes only the states that are meaningful for properties proof. A method to translate PLC programs developed in Structured Text into NuSMV models based on this representation is described and exemplified on several examples. The results, state space size and verification time, obtained with models constructed using this method are compared to those obtained with previously published methods so as to assess efficiency of the proposed representation.**

I. Introduction

Formal verification of PLC (Programmable Logic Controllers) programs thanks to model-checking tools has been addressed by many researchers ([1], [2], [3], [4], [5], [6], [7], [8]). These works have yielded formal semantics of the IEC 61131-3 standardized languages [9] as well as rules to translate PLC programs into formal models that can be taken as inputs of model-checkers such as SMV [10] or UPPAAL [11].

Despite these valuable results, it is easy to observe that model-checking is not employed daily in companies that develop PLC programs (see ([12]) for a comprehensive study of logic design practices). Automation engineers prefer to use the traditional, while being tedious and not exhaustive, simulation techniques to verify that programs they have developed fulfill the application requirements. Several reasons can be put forward to explain this situation: specifying formal properties in temporal logic or in the form of timed automata is an extremely tough task for most engineers; model-checkers provide, in case of negative proof, counterexamples that are difficult to interpret; PLC vendors do not propose commercial software able to translate automatically PLC programs into formal models, ...

All these difficulties are real and solutions must be found to overcome them, e.g. libraries of application-oriented properties, explanations of counterexamples in suitable languages, automatic translation software. Nevertheless, in our view, the main obstacle to industrial use of formal verification is combinatory explosion that occurs when dealing with large size control programs. Formal models that underlie model-checking are indeed discrete state models such as finite state machines or timed automata. Even if properties are proved symbolically, using binary decision diagrams (BDDs) for instance, existing methods produce, from industrial, large size, PLC programs, models that include too many states to be verified by the present model-checking tools. In that case, no proof can be obtained and formal verification is then useless.

The aim of the research presented in this paper is to tackle out, or at least to lessen, this problem by proposing a translation method that yields, from PLC programs, formal models far smaller than those obtained with existing methods. These novel models will include only the states that are meaningful for properties proof and then will be less sensitive to combinatory explosion. This efficient representation of PLC programs will contribute to improve scalability of model-checkers and to favor their industrial use.

This paper includes five sections. Section 2 delineates the frame of our research. The principle of the translation method is explained in section 3. Section 4 describes how efficient NuSMV models can be obtained from PLC programs developed in a standardized language thanks to this method, while section 5 presents experimental results. Prospects for extending these works are given in section 6.

PLCs (Figure 1) are automation components that receive logic input signals coming from sensors, operators or other PLCs and send logic output signals to actuators or other controllers. The control algorithms that specify the values of outputs according to the current values of inputs and the previous values of outputs are implemented within PLCs in programs written in standardized languages, such as Ladder Diagram (LD), Structured Text (ST) or Instruction List (IL). These programs run under a real-time operating system whose scheduler may be multi- or mono-task. This paper focuses only on mono-task schedulers. Given this restriction, a PLC performs a cyclic task, termed PLC cycle, that includes three steps : inputs reading, program execution, outputs updating. The period of this task may be constant (periodic scan) or may vary (cyclic scan).

II. Model-checking of logic controllers

Previous works that have been carried out to check PLC programs properties by using existing model-checkers addressed either timed ([4], [7]) or untimed ([1], [2], [3], [6], [8]) model-checking. Since our objective is to facilitate industrial use of formal verification techniques by avoiding or limiting combinatory explosion and that this objective seems more easily reachable for untimed systems, only untimed

* This work was carried out in the frame of a research project funded by Alstom Power Plant Information and Control Systems, Engineering tools Department.

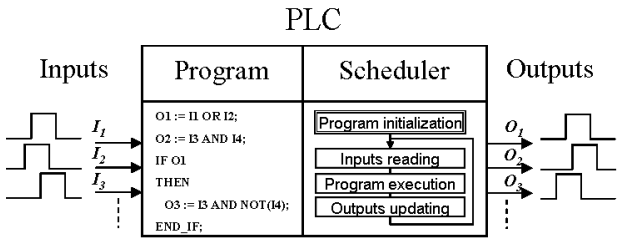

Fig. 1. PLC basic components

model-checking will be considered in this paper. In what follows, all examples of formal models will use the syntax of the NuSMV model-checker [13], though similar results would be obtained with that of other model-checkers of the same class. It matters also to point out that, given the kind of systems that are considered, periodic and cyclic tasks behave in the same fashion: PLC cycle duration is meaningless.

Several approaches have been proposed to translate a PLC program into a formal untimed model. For room reasons, only two of them will be sketched below. [14] for instance expresses the semantics of each element (contact, coil, links,...) of LD in the form of a small state automaton. The formal behavior of a given program is then obtained by composition of the different state automata that describe its elements. This method relies upon a detailed semantics of ladder diagram and can be extended to programs written in several languages, but it gives rise easily to state space explosion, even for rather small examples. A more efficient approach ([2], [6]) translates each program statement into a SMV $next$ function. Each PLC cycle is then modeled by a sequence of states, the first and last states being characterized respectively by the values of input-output variables at the input reading and output updating steps, the intermediary states by the values of these variables after execution of each statement.

Figure 2 illustrates this method on a didactic example written in ST. Thorough this paper, PLC programs examples will be given in ST. ST is a a textual language, similar to PASCAL, but tailor-made for automation engineers, for it includes statements to invoke and to use the outputs of Function Blocks (FB) such as RS (SR) - reset (set) dominant memory -, RE (FE) - rising (falling) edge. This language is advocated for the control systems of power plants that are targeted in the project. Equivalent programs in other sequentially executed languages, like programs written in IL or LD, can be obtained without difficulty.

The program presented in Figure 2 includes four statements: two assignments followed by one IF selection and one assignment. From this program, it is possible to obtain by using the previous method (translation of each statement into a SMV $next$ function) an execution trace whose part is shown on Figure 2, assuming that the values of the variables in the initial state (defined when setting up the controller) and the values of the input variables at the inputs reading steps of the first and second PLC cycles are respectively:

- Initial values of variables: $I_1 = 1$, $I_2 = 0$, $I_3 = 1$, $I_4 = 0$, $O_1 = 0$, $O_2 = 0$, $O_3 = 0$ and $O_4 = 1$
- Input variables values at the beginning of the first PLC cycle: $I_1 = 0$, $I_2 = 0$, $I_3 = 1$ and $I_4 = 1$
- Input variables values at the beginning of the second PLC cycle: $I_1 = 1$, $I_2 = 1$, $I_3 = 0$ and $I_4 = 1$

It matters to highlight that the values of input variables remain constant in all the states of one PLC cycle.

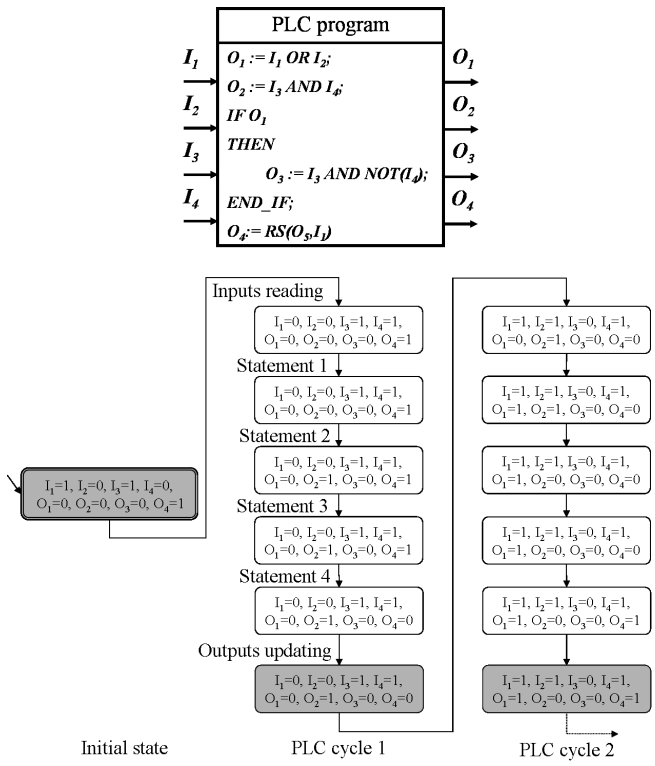

Fig. 2. A simple program and part of the resulting trace with the method presented in [6]

In addition to the formal model of the controller, model-checkers need a set of formal properties to prove. Two kinds of properties are generally considered:

- Intrinsic properties, such as absence of infinite loop, no deadlock, ..., which refer to the behavior of the controller independently of its environment;
- Extrinsic properties which refer to the behavior of inputs and outputs, e.g. commission of outputs for a given combination of inputs, always forbidden combination of outputs, allowed sequences of inputs-outputs,...

This paper focuses only on extrinsic properties. Referring to outputs behavior, these properties impact indeed directly safety and dependability of the controlled process and then are more crucial. If one of them (or several) are not satisfied, hazardous events may occur, leading to significant failures.

If focus is put on extrinsic properties verification, the two approaches described above lead to state automata with numerous states that are not meaningful. It can be seen indeed on Figure 2 that the intermediary states defined for each statement are not useful in that case; extrinsic properties

are related only to the values of input-output variables when updating the outputs, i.e. at the end of the PLC cycle. A similar reasoning may be done for the other method.

Hence efficient representation for formal verification will include only the states describing the values of input-output variables when updating outputs (shaded states in Figure 2). This representation may be obtained directly from a PLC program by applying the method whose principle is explained in the next section.

III. Method principle

A. Assumptions

In what follows it is assumed that:

- PLC programs are executed sequentially;
- only Boolean variables are used;
- internal variables may be included in the program;
- only the Boolean operators defined in IEC 61131-3 standard (NOT, AND, OR, XOR) are allowed;
- only the following statements of ST language are allowed: assignment, function and function block (FB) control statements, IF and CASE selection statements; iteration statements (FOR, WHILE, REPEAT) are forbidden;
- multiple assignments of the same variable are possible;
- Boolean FBs, such as set and reset dominant memories defined in the standard or FBs that implement application specific control rules, like actuators starting or shutting down sequences, may be included in a program.

The first two assumptions are simple and can be made for programs in ST, LD or IL. The third assumption means that a program computes the values of internal and output variables from those of input variables and of computed (internal and output) variables; this allows us to consider internal variables in the same way as outputs in what follows. The fourth and fifth ones apply only to ST programs but similar assumptions for LD or IL programs can be easily drawn up. Iterations are forbidden because they can lead to too long cycle times that do not comply with real-time requirements. The sixth assumption may be puzzling, for contrary to the usual programming rule that advocates that each variable must be assigned only once. Even if this programming rule is helpful when developing a software module from scratch, this assumption must be introduced to cope with industrial PLC programs in which it is quite usual to find multiple assignments of the same variable. Two reasons can be put forward to explain this situation. First industrial PLC programs are often developed from previous similar ones; then programs designers copy and paste parts of previous programs in the new program. This reuse practice may lead to assign one variable several times. Second a ST program may contain both normal assignments and assignments included within selection statements; this is an other reason that explains multiple assignments. As our objective is to proof properties on existing programs, without modifying them prior to verification, this specific feature must be taken into account. It will be shown below that multiple assignments do not impede to construct efficient representation.

Figure 3 outlines the translation method that has been developed to obtain efficient representation of PLC programs. As shown on this figure, this method includes two main steps: static analysis of the program and generation of the NuSMV model that describes formally the behavior of the program with regards to its inputs-outputs.

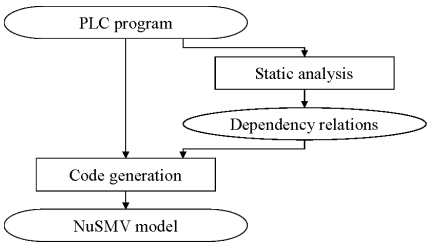

Fig. 3. Method overview

B. Static analysis

Static analysis is aiming at deriving, from the PLC program, dependency relations between variables. Starting from the initial values of input and output variables that are fixed during set up, for each PLC cycle, the values of output variables are computed either only from values of input variables or from values of input variables and values of other output variables. In the first case, the value of each output variable at the end of PLC cycle i+1 (i: positive integer) is obtained merely from values of input variables for this cycle. In the second case, computation of the value of one output variable must use the values of output variables for this cycle if the last assignment of these output variables is located upstream in the program, or the values of output variables at the previous PLC cycle (cycle i) if those variables are assigned downstream; this computation will use obviously the values of input variables for cycle i+1. Hence, the main objective of static analysis is to determine, for each output variable, whether the value of each variable involved in computation of the value of this output variable at PLC cycle i+1 is related to PLC cycle i+1 or to PLC cycle i.

Static analysis is exemplified on the program given in Figure 4. This ST program computes the values of five output variables (O_1, ..., O_5) from those of four input variables (I_1, ..., I_4) and includes only allowed statements. Some specific features of this example are to be highlighted:

- the IF statement does not specify the value of O_3 if the condition following the IF is not true; this is allowed in ST language and means that the value of O_3 remains the same when this condition is false;
- the assignment of O_4 uses the output of a RS (reset dominant memory) FB;
- one output variable (O_1) is assigned twice.

Scanning sequentially the program from top to bottom, statement by statement, static analysis yields dependency relations represented graphically in Figure 5 a). In this figure,

an arrow from variable X to variable Y means that the value of Y depends on the value of X (or that the value of X is used to compute the value of Y). Each statement gives rise to one dependency relation. For instance, the dependency relation obtained from the first statement means that the value of O_1 depends on the values of I_1 and I_2, the third relation that the value of O_3 is computed from the values of I_3, I_4, O_1, and O_3 itself (in case of false condition), the fourth relation that the value of O_4 is computed from the values of I_1, O_5, and O_4 itself (if the two inputs of a memory are false, the output stays in its previous state),.... From this first set of relations, it is then possible to build an other set of more detailed relations such as:

- there is only one dependency relation for each output variable (multiple assignments are removed);
- dependency relations are developed, if possible;
- the value of each output variable O_j (j: positive integer) at PLC cycle i+1, noted $O_{j,i+1}$, is obtained from values of input variables for this cycle, noted $I_{k,i+1}$ (k: positive integer), and from values of output variables for this cycle ($O_{j,i+1}$) or for the previous one ($O_{j,i}$).

This second set of relations is presented in Figure 5b). Only the relation coming from the latter assignment of O_1 has been kept. The first relation of the previous relations set has nevertheless permitted to obtain the final dependency relation of O_3: the value of this variable at cycle i+1 is obtained from the values of I_1, I_2, I_3, I_4 for cycle i+1 and the value of O_3 at cycle i. The computation of the value of O_4 at cycle i+1 uses the value of O_5 at cycle i for this variable is assigned after O_4 in the program whilst the value of O_5 at cycle i+1 is computed from the values of O_2 and O_4 at this same cycle because these two variables have been assigned upstream in the program.

This set of dependency relations involving the values of output variables for two successive PLC cycles permits to translate efficiently PLC programs into NuSMV models as explained in the next section.

$O_1 := I_1\ OR\ I_2;$
$O_2 := I_3\ AND\ I_4;$
$IF\ O_1$
$THEN$
$\ O_3 := I_3\ AND\ NOT(I_4);$
$END_IF;$
$O_4 := RS(O_5, I_1)$
$O_5 := O_2\ AND\ O_4;$
$O_1 := NOT(I_2\ OR\ I_4);$

Fig. 4. PLC program example

IV. Translating ST programs into NuSMV models

It is assumed in this section that the reader has a basic knowledge of the model-checker NuSMV; readers who want to know more on this proof tool can refer to [13]. To check a system, NuSMV takes as input a transition relation that specify the behavior of a Finite State Machine (FSM) which is assumed to represent this system. The transition relation of the FSM is expressed by defining the values of variables in

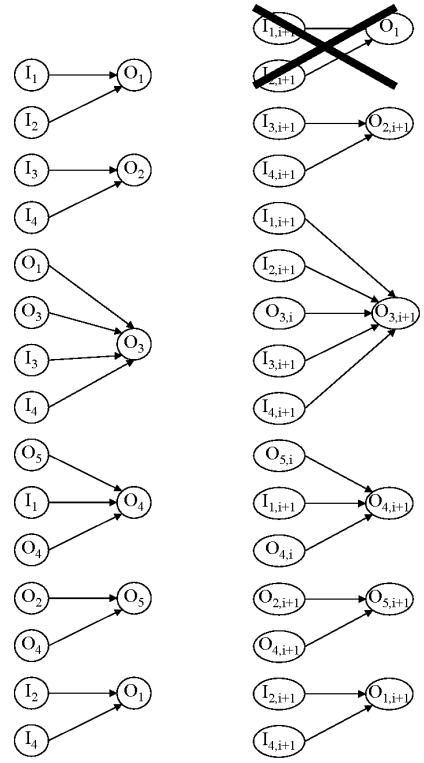

Fig. 5. Dependency relations obtained by static analysis. a) ordered intermediate relations; b) final relations

the next state (i.e. after each transition), given the values of variables in the current state (i.e. before the transition) and is described in a declarative form as a set of assignments. Each assignment defines the next value of one variable from an expression that includes operands that are values of variables in the next or in the current state, and operators. As only Boolean variables are used in this study, the only Boolean operators NOT, AND, OR, noted respectively !,& ,| will be employed below.

A. Translation algorithm

Each ST statement that gave rise to one of the final dependency relations is translated into one NuSMV assignment; then useless ST statements (assignments that are cancelled by other upstream assignments) are not translated. The set of useful statements is noted Pr in what follows. The values of the variables within one assignment are obtained from the corresponding dependency relation. If the value of a variable in this relation is that at PLC cycle i+1, then the next value of this variable will be introduced in the corresponding NuSMV assignment, using the $next$ function; if the dependency relation mentions the value at cycle i, then the corresponding NuSMV assignment will employ the current value of the variable.

Given these translation rules, the translation algorithm described Figure 6 has been developed. This algorithm yields a NuSMV model from a set of statements Pr issued from a PLC program.

```
BEGIN PLC_prog_TO_NuSMV_model(Pr)
  FOR each statement S_i of Pr:
    IF S_i is an assignment (V_i := expression_i)
    THEN
      FOR each variable V_k in expression_i:
        Replace V_k by the variable pointed out in
        the dependency graph (V_{k,i} or V_{k,i+1})
    ELIF S_i is a conditional structure (if cond; then stmt_1; else stmt_2)
      FOR each variable V_k in cond:
        Replace V_k by the variable pointed out in
        the dependency graph (V_{k,i} or V_{k,i+1})
      FOR each variable V_m assigned in S_i :
        Replace V_m assignment by:
          "case
            cond :≺assignment of V_m in
                    PLC_prog_TO_NuSMV_model(stmt_1)≻;
            !cond : ≺assignment of V_m in
                    PLC_prog_TO_NuSMV_model(stmt_2)≻;
          esac ; "
```

Fig. 6. Translation algorithm

B. Taking into account Function Blocks

If a ST assignment includes an expression involving a Boolean Function Block (FB), the behavior of this FB must be detailed in the corresponding NuSMV assignment. Hence a library of generic models describing in NuSMV syntax the behavior of the usual FBs has been developed. When translating ST assignments that include instances of FBs, instances of these generic models will be introduced into the NuSMV assignments. The RS (reset dominant memory) FB, for instance, has two inputs, noted Set and Reset, and one output Q. Its behavior is recalled below:

- If Reset is true, then Q is false;
- If Set is true and Reset false, then Q is true;
- If none of the inputs is true, then Q keeps its previous value.

This FB can be translated into the generic following NuSMV $case...esac$ structure, sequentially executed.

```
Next(Q) :=case
            Reset : 0;
            Set : 1;
            1 : Q;
          esac;
```

C. Example

```
Next(I_1) := {0,1};
Next(I_2) := {0,1};
Next(I_3) := {0,1};
Next(I_4) := {0,1};
Next(O_2) := Next(I_3) & Next(I_4);
Next(O_3) :=case
              Next(I_1) | Next(I_2) : Next(I_3) & !(Next(I_4));
              !(Next(I_1) | Next(I_2)) : O_3;
            esac;
Next(O_4) := case
              Next(I_1) : 0;
              O_5 : 1;
              1 : O_4;
            esac;
Next(O_5) := Next(O_2) & Next(O_4);
Next(O_1) :=!(Next(I_2) | Next(I_4));
```

Fig. 7. NuSMV model of the program presented in Figure 4

Using the algorithm of Figure 6, the NuSMV model presented in Figure 7 can be obtained from the program of the previous section.

It matters to emphasize that the translation algorithm does not introduce auxiliary variables, such as line counter, end of cycle, unlike the method proposed in [6]. It remains nevertheless to assess the efficiency of this representation.

V. Assessment of the representation efficiency

Several experiments have been carried out to assess efficiency of the representation proposed in this paper. To facilitate these experiments, an automatic translation program based on the method presented in the previous sections has been developed.

A. First experiment

The objective of this experiment was to compare, on the simple example of Figure 4, the sizes of the state spaces of the NuSMV models obtained with the representation proposed in [6], i.e. direct translation of each statement of the PLC program into one NuSMV assignment, and with that presented in this paper.

	Reachable states	System diameter
representation of [6]	314 out of 14336	22
proposed representation	21 out of 512	2

TABLE I

State space sizes of the program presented in Figure 4

The two NuSMV models have been first compared, using behavioral equivalence techniques, so as to verify that they behave in the same manner. This comparison gave a positive result: the sequence of outputs generated by the two models is the same whatever the sequence of inputs. Then the sizes of their state spaces have been computed, using the NuSMV forward_check function, as shown in Table I. This table contains, for each representation, the number of reachable states among the possible states, e.g. 314 among 14336 means that 314 states are really reachable among the 14336 possible, as well as the system diameter: minimum number of iterations of the NuSMV model to obtain all the reachable states. These results shows clearly that, even for a simple example, the proposed representation reduces the size of the state space by roughly one order of magnitude.

B. Second experiment

The second experiment was aiming at assessing the gains in time and in memory size, if any, due to the new representation when proving properties. This experiment has been performed using the test-bed example presented in [6]: controller of a Fischertechnik system, for which numerical results were already available. Once again two models have been developed and the same properties have been checked on both. Table II gives duration and memory consumption of the checking process for two properties. These results were obtained by using NuSMV, version 2.3.1, on a PC P4 3.2 GHz, with 1 GB of RAM, under Windows XP.

	representation of [6]	proposed representation
liveness property	5h / 526MB	2s / 8MB
safety property	20min / 200MB	2s / 8MB

TABLE II

TIME AND MEMORY REQUIRED FOR PROPERTIES VERIFICATION

This experiment shows that the proposed representation reduces significantly the verification time and the memory consumption. The ratio between the verification times obtained with the two representations, for instance, varies between 9000 and 600, depending on the property. Similar results are obtained with the other properties.

C. Third experiment

This third experiment has been performed with industrial programs developed for the control of a thermal power plant. The control system of this plant comprises 175 PLCs connected by networks. All the programs running on these PLCs have been translated as explained previously. The objective of this experiment was merely to assess the maximum, medium and minimum sizes of the state spaces of the models obtained from this set of industrial programs when using the proposed representation. These values are given on the fourth line of Table III. Even if the sizes of the state spaces are very different, this experiment shows clearly the possibility of translating real PLC programs without combinatory explosion. Moreover these state spaces can be explored by the model-checker in a reasonable time, a mandatory condition for checking properties; only 8 seconds are necessary indeed to explore all the state spaces of these programs. A secondary result is given at the last line of this table; the translation time, time necessary to obtain from the set of programs a set of NuSMV models in the presented representation complies with engineering constraints; translation of one PLC program into one NuSMV model will not slow down PLC program design process.

Number of programs	175
Output variables	max:47 min:1 sum:1822
Input variables	max:50 min:2 sum:2329
State space size of each program	max:8.10^{28} min: 10^5 mean:5.10^{26}
Strutation time of all state spaces	8 sec
Whole time for translation	50 sec

TABLE III

RESULTS FOR A SET OF INDUSTRIAL PROGRAMS

Even if it is not possible to obtain from these three experiments definitive numerical conclusions, such as state space reduction rate, verification time improvement ratio, ... they have allowed to illustrate the benefits of the proposed representation on a large concrete example, coming from industry.

VI. CONCLUSION

The representation of PLC programs proposed in this paper can contribute to favor dissemination of model-checking techniques, for it enables to lessen strongly state space explosion problems and to reduce verification time. The examples given in the paper were written in ST language. Nevertheless programs written in LD or in IL languages can be represented in the same manner; the principle of the translation method is the same, only the translation rules of statements are to be modified.

Ongoing works concern an extension of this representation to take into account integer variables and the development of a similar representation for timed model-checking.

REFERENCES

[1] I. Moon, "Modeling programmable logic controllers for logic verification," in *Control Systems Magazine, IEEE*. IEEE Comp. Soc. Press, 1994, pp. 53–59.

[2] M. Rausch and B. Krogh, "Formal verification of PLC programs," in *Proc. of American Control Conference*, June 1998, pp. 234–238.

[3] R. Huuck, B. Lukoschus, and N. Bauer, "A model-checking approach to safe SFCs," in *Proc. of CESA 2003*, July 2003.

[4] B. Zoubek, "Automatic verification of temporal and timed properties of control programs," Ph.D. dissertation, University of Birmingham, 2004.

[5] G. Frey and L. Litz, "Formal methods in PLC programming," in *Proc. of the IEEE SMC 2000*, October 2000, pp. 2431–2436.

[6] O. de Smet and O. Rossi, "Verification of a controller for a flexible manufacturing line written in ladder diagram via model-checking," in *American Control Conference, ACC'02*, May 2002, pp. 4147–4152.

[7] H. Bel Mokadem, B. Bérard, V. Gourcuff, J.-M. Roussel, and O. de Smet, "Verification of a timed multitask system with Uppaal," in *Proc. of ETFA'05*. Catania, Italy: IEEE Industrial Electronics Society, Sept. 2005, pp. 347–354.

[8] F. Jiménez-Fraustro and É. Rutten, "A synchronous model of IEC 61131 PLC languages in SIGNAL." in *ECRTS*, 2001, pp. 135–142.

[9] *IEC Standard 61131-3 : Programmable controllers - Part 3*, IEC (International Electrotechnical Commission), 1993.

[10] K. L. McMillan, *The SMV Language*, Cadence Berkeley Labs, http://www-cad.eecs.berkeley.edu/~kenmcmil/language.ps.

[11] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL – a tool suite for automatic verification of real-time systems," in *Proc. Workshop Hybrid Systems III: Verification and Control, New Brunswick, NJ, USA, Oct. 1995*, ser. Lecture Notes in Computer Science, vol. 1066. Springer, 1996, pp. 232–243.

[12] M. R. Lucas and D. M. Tilbury, "A study of current logic design practices in the automotive manufacturing industry," *Int. J. Hum.-Comput. Stud.*, vol. 59, no. 5, pp. 725–753, 2003.

[13] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking," in *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, ser. LNCS, vol. 2004. Copenhagen, Denmark: Springer, July 2002.

[14] O. Rossi, "Validation formelle de programmes ladder diagram pour automates programmables industriels (formal verification of PLC program written in ladder diagram)," Ph.D. dissertation, ENS de Cachan, 2003.