Cloud-Based Framework for Practical Model-Checking of Industrial Automation Applications

Sandeep Patil, Dmitrii Drozdov, Victor Dubinin, Valeriy Vyatkin

Cloud-Based Framework for Practical Model-Checking of Industrial Automation Applications

Sandeep Patil[1], Dmitrii Drozdov[1,2], Victor Dubinin[2], Valeriy Vyatkin[1,3]

[1]Luleå University of Technology, Luleå, Sweden {sandeep.patil@ltu.se, dmitrii.drozdov@ltu.se, vyatkin@ieee.org}
[2]Penza State University, Penza, Russia {victor_n_dubinin@yahoo.com}
[3]Aalto University, Helsinki, Finland

Abstract. In this paper we address practical aspects of applying the model-checking method for industrial automation systems verification. Several measures are proposed to cope with the high computational complexity of model-checking. To improve scalability of the method, cloud-based verification tools infrastructure is used. Besides, closed-loop plant controller modelling and synchronization of transitions in the SMV (input language for symbolic model checking) model aim at complexity reduction. The state explosion problem is additionally dealt with by using an abstraction of the model of the plant with net-condition event systems, which is then translated to SMV. In addition, bounded model-checking is applied, which helps to achieve results in cases when the state space is too high. The paper concludes with comparison of performance for different complexity reduction methods.

Keywords: Formal Verification, Closed-Loop Modelling, Model-checking, SMV, NCES, Industrial Automation, IEC 61499.

1 Introduction

Formal verification is an act of proving or disproving an algorithm with respect to some specification or property and model-checking is one such formal verification approach introduced in early 1980s by Clarke and Emerson [1, 2]. A model-checker generates state space of the model that includes all or some states and transitions. Each path in the state space corresponds to one system's run or a single test case. Model-checking enables the unsupervised automatic verification process of a system by generating the state space and identifies system failure via counterexamples (a scenario (state trace) that breaks the safety property of the system). Properties to be verified are specified using many methods, including temporal logic, automata, etc. While model-checking is computationally resource hungry, it has been successfully used in other adjacent areas of computer systems engineering, such as hardware design, proving its ability to handle problems of reasonably large complexity [3, 4]. This suggests that it can be also applied in the industrial automation domain, and there have been impressive number of research works towards this goal [5, 6]. For safety-critical applications, formal verification is one of the most efficient ways to prove system's correctness.

2 Cloud-based Model-Checking Infrastructure

Verification tasks consume many resources and verification process can run for days and block all important personal computer resources. In this section, we present how we could run verification remotely on a service such as Amazon Elastic Compute Cloud that grows according to the need. A verification problem usually contains specifications that are to be evaluated to prove or disprove many properties and each property checking can run as a separate process. Hence the option to perform verification tasks in the cloud instead of running them on an own computer provides performance related improvements. The cloud-based solution takes advantage of highly scalable parallel computations and more cost-efficiency in comparison to supporting own computation infrastructure.

Fig. 1 shows possible decomposition of full verification with Bounded Model Checking (BMC). For each single bound, the NuSMV [7] verifier can generate a separate SAT-problem [7]. And all these SAT problems can be solved in parallel tasks (on many Amazon Cloud instances for example).

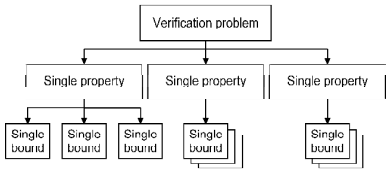


Fig. 1. Dividing the verification problem across multiple instances.

In this work it is assumed that controllers are presented in terms of the function blocks of IEC 61499 standard that generalizes traditional programming paradigms of industrial automation for the case of component-based distributed systems.

For the function block models, memory consumption grows linearly for a single bound verification with NuSMV and this behaviour provides highly predictable resource needs. Reconfigurable clouds such as example Amazon EC2, enables us to tune cloud resources for specific model and reach high cost-efficiency.

3 Model Reduction with Plant Abstraction

In model-checking one of the most common problems is the state space explosion[8], which is tackled by various techniques [9]. In this section we present an abstraction technique in order to reduce the state space of the formal model that we use. This abstraction is built on top the closed-loop modelling previously proposed in model-checking of function block systems [10].

3.1 Plant Model Abstraction using Net Condition-Event Systems (NCES)

In the closed-loop framework, both controller and plant are modelled with formalism, and then the combined model is exposed to model-checking. The controller model is

automatically generated from the function block source code, using the function block (FB) operational semantics [11]. Therefore abstractions cannot be applied to the controller's model without loss of important execution aspects, hence we abstract on the model of the plant.

The control signals actuating the plant can be both value (level) and event (edge) signals. The formalism of NCES [12, 13] has sufficient expressiveness to represent them.

In works [10, 14] modelling and verification of closed-loop automation systems using NCES was described. In work [15] the methods to model Petri net in SMV [16] were proposed. Using first and second methods in combination with event signal transfer rules from FB operational semantics, hierarchical NCES models can be transformed into SMV.

The NCES model is wrapped into FB SMV module interface and contains input transition-place pairs (Fig. 2 (a)) which are designed such that all input transitions are fired synchronously. If an activation event occurs, transition rewrites place marking regard to 'data input' value, for simple plant models with Boolean inputs, considered in this paper, the place becomes marked if 'data input' value is TRUE, otherwise the place becomes unmarked.

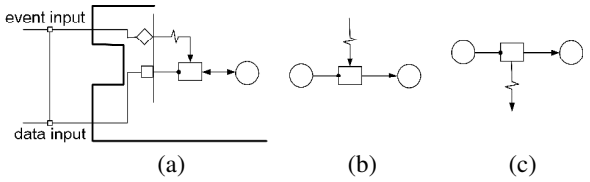


Fig. 2. (a) Synchronous transition-place pair and (b, c) typical NCES transitions

Fig. 2(b) shows a typical event activated transition (event consumer). The simple Petri net transition SMV module [15] can be transformed into NCES transition with event trigger by adding an event in 'enable' rule and simple unconditional event reset statement.

This approach mainly corresponds to events in FB operational semantics. The code snippet below shows the SMV module with the changes, line 2 shows the enable rule and the '*next()*' statements show setting/resetting rules.

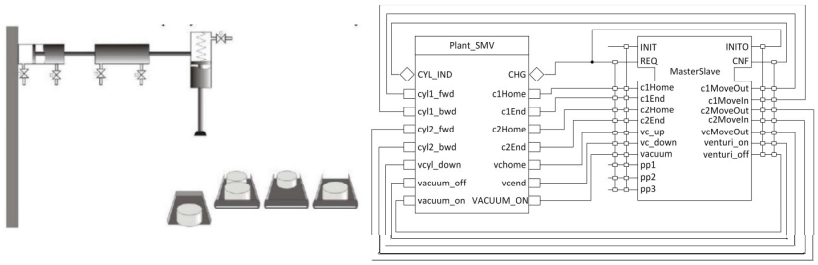
```
MODULE pnpVacuumOn(event_in, input, output)
DEFINE enabled := input & event_in;
ASSIGN
next(input) := case
  enabled : FALSE;
  TRUE :  input;
esac;
(*… Output goes here … *)
next(event_in) := FALSE;
```

Fig. 2 (c) shows a typical event emitting transition. Event signal transmission is performed through signal buffers (SMV variables), which are set as 'true' when event emitting module fires and reset as 'false' every time an event consumer module runs. The code snippet below shows the SMV module exemplifies the setting and resetting of event emission.

```
MODULE pnpVacuumOn(input, output, event_out)
DEFINE enabled := input;
ASSIGN
next(input) := case
        enabled : FALSE;
        TRUE :  input;
esac;
(*… Output goes here … *)
next(event_out) := case
          enabled : TRUE;
          TRUE :  event_out;
esac;
```

3.2 Example: A 3 Cylinder Pick-n-Place Manipulator

Fig. 3 (a) shows the 3-cylinder Pick-and-Place manipulator case study. It consists of 2 vertical and 1 horizontal cylinders and a suction unit (vacuum). Each cylinder has two sensors indicating two end positions of the piston. Full model verification with NCES was described in works [10, 14], in this paper we propose a way to verify full operational semantics-based controller model [11] with plant abstraction.



(a) (b)

Fig. 3. 3-cylinder PnP manipulator and its closed-loop model (with NCES plant) schema

Fig. 3(b) shows the closed-loop model schema with MasterSlave controller[17] and the NCES plant model. At the initial state, the input 'INIT' has value 'TRUE', which corresponds to the occurrence of event 'INIT' in the beginning of execution. Event output 'INITO' is connected to input 'REQ'. This connection provides an entry point for system's main execution path, which may be described as a cycle of the occurrence of events

REQ -> CNF -> CYL_IND -> CHG -> REQ ->…

Also, input pp1 is 'true' and pp2 and pp3 are 'false' (workpiece is present in place 1 and places 2 and 3 are empty) for model testing. But in real systems checking these variables can be left "free" to verify all possible system states, or they can be connected to environment emulation module.

The simplest abstract model of the cylinder consists of only two states: 'home' and 'end' and two transition between these states. Transition emits event 'CHG' when it fires. Vacuum model contains only two states as well.

Fig. 4(a) shows simple NCES models for vacuum and cylinder modules. The vacuum module has two inputs 'vacuum_on' and 'vacuum_off' and two outputs:

'vacState' and event 'CHG'. Two places 'On' and 'Off' represent simple vacuum states and transitions between this states emit event 'CHG' when fire. Cylinder model has two inputs and two states as well, but transitions with inhibitor arcs provide operation regard to rules:

$$ToHome.enabled = fwd \,\&\, \overline{bwd};$$
$$ToEnd.enabled = \overline{fwd} \,\&\, bwd;$$

Fig. 4(b) shows the plant model with synchronous transitions inside the FB interface (wrapper). Inputs of NCES cylinders and vacuum are connected to synchronous transition-place pairs and outputs are merged with wrapper's outputs (in this case, event signal buffers reset is managed by wrapper and "outer" SMV module regard to FB operational semantics).

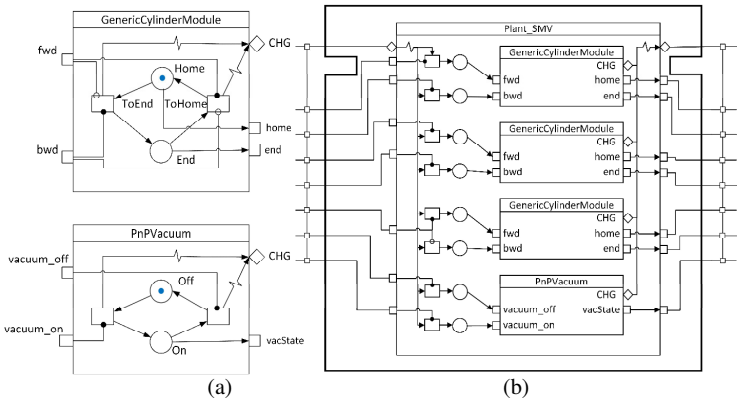


Fig. 4. (a) Vacuum and cylinder NCES-models and (b) Plant model with FB wrapper

4 Synchronous Function Block SMV Models

4.1 SMV Processes

Most of SMV language implementations support asynchronous module instances, implemented with keyword 'process' [16]. Usage of asynchronous processes provides simple representation or concurrent tasks but analysis of all possible execution orders together with increasing number of processes causes exponential state space growth (so-called state space combinatorial explosion). This problem affects resource consumption, for example memory and verification time.

FB application SMV model, based on FB operational semantics, proposed in work [11] uses FB instance mapping to asynchronous SMV processes. This approach provides intuitively clear FB types and instances mapping and allows avoiding the problem with access of multiple modules to one signal buffer. It is also the simplest way to model asynchronous execution discipline in SMV. Fig. 5 shows access of two modules to event signal buffer. Value 'TRUE' of the buffer SMV variable means the event occurred and has not been read yet. So one module "puts" an event to the buffer

(set TRUE value to the buffer SMV variable) and another module receives event (reads the variable) and resets SMV variable value to FALSE.

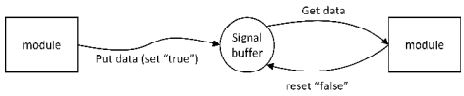
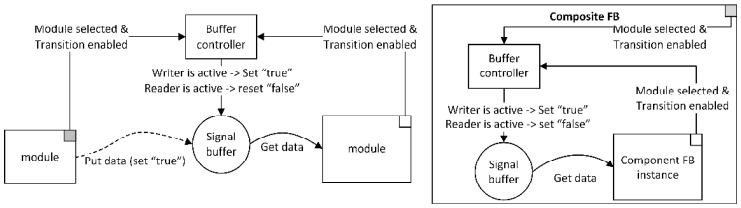


Fig. 5. Access to shared signal buffer

4.3 Synchronous Module Instances with External Buffer Controller

The approach to solve the multiple access problems with synchronous modules is based on external buffer controller, which contains all assignments of the shared buffer. (b)
Fig. 6(a) shows the controller that uses module selector variable to select one of many assignment rules, collected from accessing modules.



(a) (b)

Fig. 6. (a)External buffer controller and (b) Buffer controller in composite FB

4.4 Synchronous FB Modules with Buffer Controller in SMV

In the function block SMV model, the execution order is fully determined by dispatchers. The problem of concurrent access to SMV variable occurs in event signal buffers, which are accessible from composite FB and its components. In this case, the schema shown in (b)
Fig. 6(a) can be modified and external buffer controller can be placed in a composite FB as it is shown in (b)
Fig. 6(b).

A composite FB model contains buffer SMV variables and it should contain buffer controllers. Composite FB SMV module and each asynchronous component instance contain own 'next' expressions to change shared event signal buffers (setting of output and resetting of input events). For synchronous modules this expressions from all component instances can be merged in composite FB and placed either in separated buffer controller modules, or in composite FB module definition.

signal_reset_rule and *signal_set_rule* in component FBs are declared with DEFINE statements. Alpha and beta rules should be defined in components and their 'next' statements should be merged into composite FB dispatcher.

For component event inputs.

```
next(event_signal):= case
  signal_set_rule : TRUE;

component.signal_reset_rule
: FALSE;
  TRUE : event_signal;

esac;
```

For component event outputs.

```
next(event_signal):= case
  component.signal_set_rule
: TRUE;
  module_is_active : FALSE;
-- unconditional reset
  TRUE : event_signal;

esac;
```

5 Results

Table 1 shows comparison of RAM usage and verification time for full PnP model and reduced with the first approach (described in section 3). Table 2 shows comparison of RAM usage and verification time for bounded model checking with single bounds for PnP model with asynchronous modules and synchronous ones.

Table 1: Model checking in Cadence SMV

Model	Time	RAM
Full PnP model	–	> 2GB*
Reduced PnP model	294 s	1082 MB

* On Win32 Cadence SMV running under Windows 7 x64, reaching of 2GB memory limit cause program failure, on Windows x32 versions, verification process stuck for hours without any results.

Table 2: Bounded model checking with the second approach

Single bound value	PnP model with async. instances		PnP model with sync. instances	
	Time	RAM	Time	RAM
20	42.93 s	2134 MB	20.37 s	1875 MB
50	> 12 hours	~7 GB	58.15 s	4721 MB
70	> 12 hours	~ 11 GB	73.18 s	5689 MB

6 Conclusion

Proposed modification of function blocks SMV models with synchronous modules allows for reduction of BMC verification time to reasonable limits. And the abstraction technique, using NCES to represent plant model, allows us to reduce model complexity when it necessary to keep within memory limits. Both, these techniques allow us to apply model-checking for verification of IEC 61499 closed-loop systems, and make this approach cost-efficient with cloud technologies.

Acknowledgments. This work was partially supported by the Federal Targeted Programme "Research and development in priority areas of elaboration of STC of Russia for 2014-2020" (agreement of 19.06.2014 № 14.574.21.0045), and by Luleå Tekniska Universitet, grants 381119, 381940 and 381121.

References

1. E. M. Clarke and E. A. Emerson, "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic," Logic of Programs Workshop, 1982.
2. E. A. Emerson and E. Clarke, "Characterizing correctness properties of parallel programs using fixpoints," in *Automata, Languages and Programming*. vol. 85, J. de Bakker and J. van Leeuwen, Eds., ed: Springer Berlin Heidelberg, 1980, pp. 169-181.
3. L. Fix, "Fifteen Years of Formal Property Verification in Intel," in *25 Years of Model Checking*, G. Orna and V. Helmut, Eds., ed: Springer-Verlag, 2008, pp. 139-144.
4. C. Kern and M. R. Greenstreet, "Formal verification in hardware design: a survey," *ACM Trans. Des. Autom. Electron. Syst.,* vol. 4, pp. 123-193, 1999.
5. H.-M. Hanisch, M. Hirsch, D. Missal, S. Preuße, and C. Gerber, "One Decade of IEC 61499 Modeling and Verification-Results and Open Issues," in *13th IFAC Symposium on Information Control Problems in Manufacturing*, V.A. Trapeznikov Institute of Control Sciences, Russia, 2009.
6. V. Vyatkin and H. M. Hanisch, "Formal modeling and verification in the software engineering framework of IEC 61499: a way to self-verifying systems," in *Emerging Technologies and Factory Automation, 2001. Proceedings. 2001 8th IEEE International Conference on*, 2001, pp. 113-118 vol.2.
7. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri*, et al.*, "NuSMV 2: An OpenSource Tool for Symbolic Model Checking," in *Computer Aided Verification*. vol. 2404, E. Brinksma and K. Larsen, Eds., ed: Springer Berlin / Heidelberg, 2002, pp. 241-268.
8. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Progress on the State Explosion Problem in Model Checking," in *Informatics*. vol. 2000, R. Wilhelm, Ed., ed: Springer Berlin Heidelberg, 2001, pp. 176-194.
9. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-Guided Abstraction Refinement," in *Computer Aided Verification*. vol. 1855, E. A. Emerson and A. Sistla, Eds., ed: Springer Berlin Heidelberg, 2000, pp. 154-169.
10. S. Patil, S. Bhadra, and V. Vyatkin, "Closed-loop formal verification framework with non-determinism, configurable by meta-modelling," in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, 2011, pp. 3770-3775.
11. S. Patil, V. Dubinin, C. Pang, and V. Vyatkin, "Neutralizing Semantic Ambiguities of Function Block Architecture by Modeling with ASM " in *9th International Andrei Ershov Memorial Conference, PSI 2014*, Peterhof, St. Petersburg, Russia, 2014.
12. H.-M. Hanisch and A. Lüder, "Modular modeling of closed-loop systems," in *Proc of Colloquium on Petri Net Technologies for Modeling Communication Based Systems*, ed Berlin, Germany, 2000, pp. 103-126.
13. L. Pinzon, M. A. Jafari, H. M. Hanisch, and Z. Peng, "Modeling admissible behavior using event signals," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on,* vol. 34, pp. 1435-1448, 2004.
14. S. Patil, V. Vyatkin, and M. Sorouri, "Formal verification of Intelligent Mechatronic Systems with decentralized control logic," in *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*, 2012, pp. 1-7.
15. G. Wimmel, "A BDD-based Model Checker for the PEP Tool," *Major Individual Project Report, Dept,* 1997.
16. (4th March). *Cadence SMV Model Checker*. Available: http://www.kenmcmil.com/smv.html
17. M. Sorouri, S. Patil, and V. Vyatkin, "Distributed control patterns for intelligent mechatronic systems," in *Industrial Informatics (INDIN), 2012 10th IEEE International Conference on*, 2012, pp. 259-264.