Using plant model features to generate reduced test cases for programmable controllers

Canlong Ma, Julien Provost

Technical University of Munich, Safe Embedded Systems,
85748 Garching bei München, Germany
(e-mail: {ma, provost}@ses.mw.tum.de).

Abstract: Complete conformance testing is a model-based test technique for programmable controllers. It checks whether an implementation conforms to its specifications with regard to all possible combinations of input signals, which is useful for small scale and safety critical systems. However, the *state space explosion* issue limits its application to large scale systems. This paper presents a method for reducing state space in generation of test cases by utilizing not only specification models but also features extracted from plant models. The application on a benchmark case study shows that the number of test cases is reduced significantly.

Keywords: discrete event system, programmable controller, testing, validation

1. INTRODUCTION

Industrial automation systems such as manufacturing are facing challenges of rising complexity and flexibility. This fact leads to an urgent demand of qualitative testing with convincing reliability and high efficiency (Rösch et al. (2015)). Controllers are key parts in an automation system as they make orders to actuators based on the signals received from sensors. An important task in testing of an industrial automation system is therefore the testing of its controllers.

Programmable controllers with cyclic execution mode are widely used in industry since they are robust enough to endure in industrial environment, have standard programming languages, and meet the 'hard' real-time requirement. In the design phase, specifications are made according to users' requirements. Then, based on the specifications, executable programs are implemented on programmable controllers. In the execution phase, when a programmable controller is turned on, it runs in cycles: reading values of input signals, executing implemented programs, updating values of output signals.

Conformance testing is a model-based test technique that aims to check whether an implementation performs the same behavior with regard to its specifications (Provost et al. (2014)). It is recommended by a series of international certification standards such as *IEC 61850-10* and *IEC 60880*. Conformance testing on a programmable controller consists of three phases: test case generation, test execution, and result verdict.

In the classic strategy, a complete set of test cases is generated from specification models directly and considers all possible combinations of input signals (Provost et al. (2011)). It is worth noting that plant behaviors are usually not considered because they often do not even exist. The few exceptions are: Supervisory control theory (Ramadge and Wonham (1987)), which is unfortunately not receiving the expected attention in the industry up to now; formal verification techniques, some of which use plant models when checking logical properties (Frey and Litz (2000), Machado et al. (2006)); and hardware-in-the-loop test benches (Gu et al. (2007)), which permits to simulate the future plant in a test bench during test execution. Yet, to the best of our knowledge, plant models have not been considered for the generation of test sequences.

Complete conformance testing is advantageous for a *system under test* (SUT) that is safety critical, since it, by design, covers the whole behavior defined by the specification. However, it suffers two issues.

The first issue occurs during test execution and result verdict: erroneous test verdicts might happen due to incorrect detection of synchronous input changes by a programmable controller under test. This has been formally described as single-input-change (SIC) testability issue in Provost et al. (2014). To fulfill the requirement of full SIC-testability in testing programmable controllers, a design-to-test (DTT) approach was proposed in Ma and Provost (2015), which also improves controllability and observability performance in testing additionally. In brief, the DTT approach modifies specification models to improve the testability of their implementations with limited design and testing overhead. For more details, a software toolbox has been developed for the DTT approach (Ma and Provost (2016)).

The second issue is the *state space explosion* issue. With complete testing, the number of test cases grows exponentially with the number of inputs of a SUT, which severely restricts its application to large scale systems.

The method proposed in this paper aims to solve this second issue. The core idea is to use not only specification but also plant models in generation a smaller set of test

cases. With additional information from plant models as well as interactions between specification and plant models, the reduced test cases focus mainly on the nominal behavior of the system, which is more realistic for large scale systems. Moreover, testing the nominal behavior of a SUT is a prerequisite to the test of its faulty behavior; if its nominal behavior is not correct, testing its faulty behavior would be meaningless. Also, for large scale systems, only a subset of the whole behavior is critical and requires a complete testing.

A big advantage of the proposed method is that it does not necessarily require highly detailed or full plant models. Any piece of plant knowledge can contribute to the reduction of test cases. Therefore, in practice, the obstacle of the *state space explosion* issue in conformance testing can be diminished.

The paper is structured as follows: Section 2 presents the formalism of finite state machines used in specification and plant models. Section 3 presents different description methods of signal relations and the utilization of plant models in test case generation. A benchmark case study is illustrated in section 4. Finally, a discussion of this work is given in the last section.

2. BACKGROUND

2.1 Specification model: Communicating Moore machine with Boolean signals

In this paper, specifications of a system are modeled as a set of Moore machines, a type of finite state machine (FSM), which can communicate with each other via internal variables, adapted from Lee et al. (1996).

Due to simplicity reason and a wide range of applications, Boolean signals are used as inputs and outputs in the illustration of the proposed method. However, the method can also be applicable to general digital signals with a few adaptations.

An important thing to keep in mind is that, compared to event based models, signal based models do not restrict only one change of input values at once (Provost et al. (2011)).

A communicating Moore machine extended with Boolean signals is defined by a 7-tuple $(S, s_{init}, I, C, O, \delta, \lambda)$, where:

- S is a finite set of states
- s_{init} is the initial state, $s_{init} \in S$
- I is a finite set of Boolean input signals
- C is a finite set of internal Boolean communicating variables
- O is a finite set of Boolean output signals
- $\delta : S \times 2^{I+C} \rightarrow S$ is the transition function that maps the current state and Boolean expression, which is made up of input signals and communicating variables, to the next state
- $\lambda : S \rightarrow 2^{O}$ is the output function that maps the states to their corresponding output signals

A Boolean expression used in a transition is denoted as a 'transition guard'. A transition is fired when its source state is active and its guard is evaluated as '1' (i.e. *True*).

Moore machines are also represented in graphical form in this paper. A simple example is given in Fig. 1.

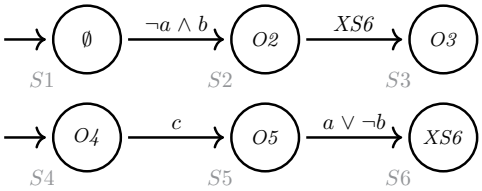


Fig. 1. A simple example of a Moore machine with Boolean signals

A state s is drawn as a circle or a rounded rectangle. A transition δ is represented by an oriented arc with its guard, e.g. $\neg a \wedge b$ for the transition from $S1$ to $S2$.

A state can either have an externally observable action, e.g. *O2* in $S2$, or no observable action, e.g. $\emptyset$ in $S1$. Additionally, a state can also be given an internal communicating variable, e.g. *XS6* in $S6$, which can be used in Boolean transition guards. For example, when the state $S6$ is activated, *XS6* is then assigned the value '1'. If $S2$ is active at the same time, then the transition from $S2$ to $S3$ can be fired.

In hierarchical modeling, a state can contain other states which are called sub-states (Girault et al. (1999)).

2.2 Synchronous composition of individual specification models

When modeling a complex industrial process, it is convenient to build several simple individual models and compose them, instead of directly constructing a large monolithic model.

For composition of FSM models, a significant number of theory research has been done since many years. Practical tools such as Teloco (Provost et al. (2011)) are available to obtain a composed model from individual models.

The formalism of a composed machine is similar to an individual Moore machine. The main differences are:

- In a composed model, a location represents a combination of states from the individual models.
- A transition function between locations is named an 'evolution'.

It is worth mentioning that with Teloco, the composed model contains only stable locations, i.e. locations where only a change in the input values can trigger a change of locations (Provost et al. (2011)).

2.3 Plant model as finite state machine

In this paper, plants can also be modeled as finite state machines with Boolean signals to describe dependency relations between signals.

The formalism of a plant model is similar to a specification model except two terms:

- $\lambda : S \rightarrow 2^{I}$: *inputs* of specification models are used as *outputs* in plant models

- $\delta : S \times 2^{I+O} \rightarrow S$: both *inputs* and *outputs* of specification models can be used in the *guards* in plant models

A simple example of a plant model is given in Fig. 2, which interacts with the second specification model in Fig. 1.

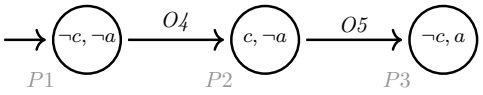


Fig. 2. A simple example of a plant model as FSM

The signal c cannot be *True* until *O4* is activated. A similar relation exists between a and *O5*. In addition, a and c cannot be *True* at the same time according to this plant model.

3. PLANT MODEL IN TEST CASE GENERATION

3.1 Complete vs. reduced testing case generation

As presented in the both figures of Fig. 3, a conformance testing unit consists of three phases: test case generation, test execution, and result verdict.

Test cases made up of input and output sequences are generated from specification models. In each single cycle of the test execution, a programmable controller under test is solicited with the input sequence; after execution of the implemented program, the controller updates its output sequence.

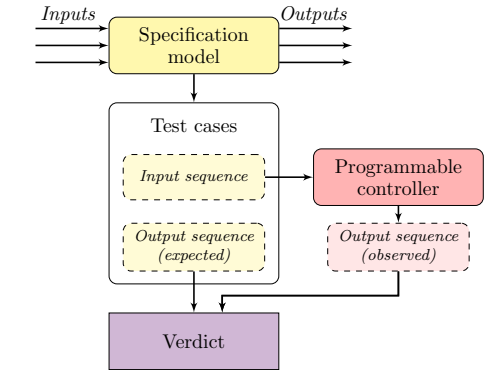
The output sequence observed from the controller is then compared with the expected output sequence from specifications. This is the verdict phase.

Considering Fig. 3(a), a complete set of test cases should contain all the possible combinations of input signals from all states derived from specification models, which grows exponentially with the number of inputs of a SUT. A complete testing is necessary and computable for critical systems with limited inputs. However, it is intractable for large scale systems, as explained in Sec. 1.

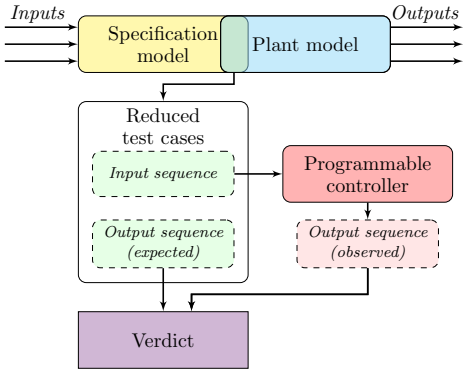
The goal of the proposed method in this paper is to reduce the state space in generating test sequences by utilizing features extracted from plant models. Its framework is presented in Fig. 3(b). The consistency of a combination of input values is then checked according to plant model before being generated as a test case. For example, in a container in case a sensor of high level gives a value *True* and a sensor of low level gives a value *False*, this is an inconsistent combination because it cannot happen in a nominal model. Thus, this combination is removed from the set of test cases.

The proposed method contains not only models of specifications, but also models of plants in the generation of test cases. Thus, it differs from 'Simulation in the loop testing', which only involves specification models (Jeon et al. (2010), Lee and Drury (2013)).

It is worth noting that any piece of information of signal relations contributes to the reduction; the method does not necessarily require full knowledge of the plant, which makes the method more practical in industry.



(a) Complete testing



(b) Testing with reduced test cases

Fig. 3. Two conformance testing frameworks for programmable controllers

3.2 Description of signal relations

Signal relations can be expressed through different formal and informal languages. In the following parts three methods are used to describe two basic types of signal relations that are presented in Fig. 4: $a - b$ and $c - d$.

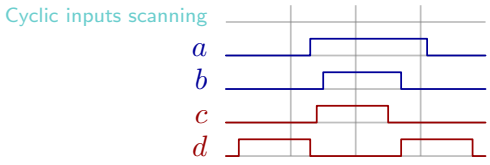


Fig. 4. Two basic types of signal relations

Natural language With natural language, the two types of signal relations can be organized as follows:

- Signal b can only be *True*, when a is *True*, i.e. a is premise of b
- Signal c and d are mutually exclusive, i.e. at the same time, only one of c and d can be *True*

Temporal logic Linear temporal logic (LTL) and computation tree logic (CTL) are two popular forms of temporal logic, a formal verification language. They can also be applied to depict the signal relations:

- *LTL*: $G(\neg a \rightarrow \neg b)$
  CTL: $AG(\neg a \rightarrow \neg b)$
- *LTL*: $G\neg(c \wedge d)$
  CTL: $AG\neg(c \wedge d)$

Finite state machine The modeling language FSM can also be used to formalize the signal relations, as presented in Fig. 5.

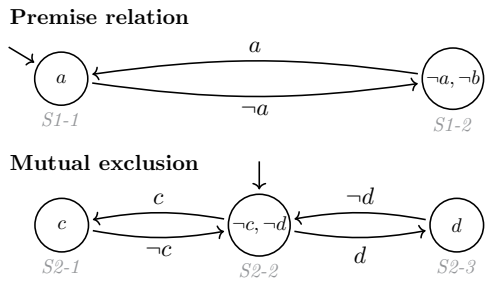


Fig. 5. Representation of the premise relation and mutual exclusion of signals with FSM [1]

In fact, the two basic types of signal relations, premise and mutual exclusion, can be combined to construct complex signal relations when involving several signals.

For example, signals in a system can have such behavior: if a is *True* and remains *True*, and b becomes *True*, then c can be *True*; once a becomes *False*, c turns to *False* as well. With CTL the signal relations can be expressed as: $AG\big((\neg a \rightarrow \neg c) \wedge (a \wedge b \rightarrow AFc)\big)$. The same signal relations can be modeled as FSM, as presented in Fig. 6.

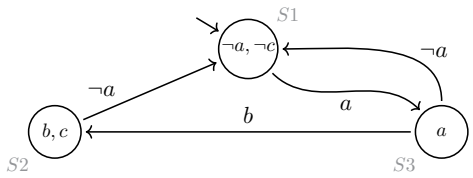


Fig. 6. A simple example of multiple signal relations

In brief, natural language is well capable to handle small scale systems with a limited number of signals. For large scale systems, it is recommended to use one of the formal methods, i.e. temporal logic and/or finite state machines.

3.3 Composition of the specification and plant models

After extracting the signal features, the next step is to compose the specification and plant models.

For FSM form of plant models, methods of synchronous composition introduced in Sec. 2.2 can be applied.

For plant models as temporal logic, methods have been developed to transform the formulas to finite state machines (Bloem et al. (2012)). Thus, the specification and plant models can also be composed.

For plant models as natural languages, patterns could be used to help users transform them into formal languages (Fantechi et al. (1994), Campos et al. (2008)).

[1] Signals values can be freely assigned if they do not appear in the initial state, i.e. the output of *S1-1* can either be (a, b) or $(a, \neg b)$. Once set, the value of a signal remains unchanged until it is deliberately modified, i.e. the explicit output of *S2-1* is $(c, \neg d)$.

4. APPLICATION ON A CASE STUDY

4.1 Benchmark case study

In this paper, an automatic weighing-mixing system adapted from IEC 60848 (2011) is used as a benchmark to illustrate the proposed method. Formally defined requirements of the benchmark have been transformed into FSM (Ma and Provost (2015)), some of which are used in this paper.

The system sketch is presented in Fig. 7. Its input and output signals are listed in Tab. 1, respectively.

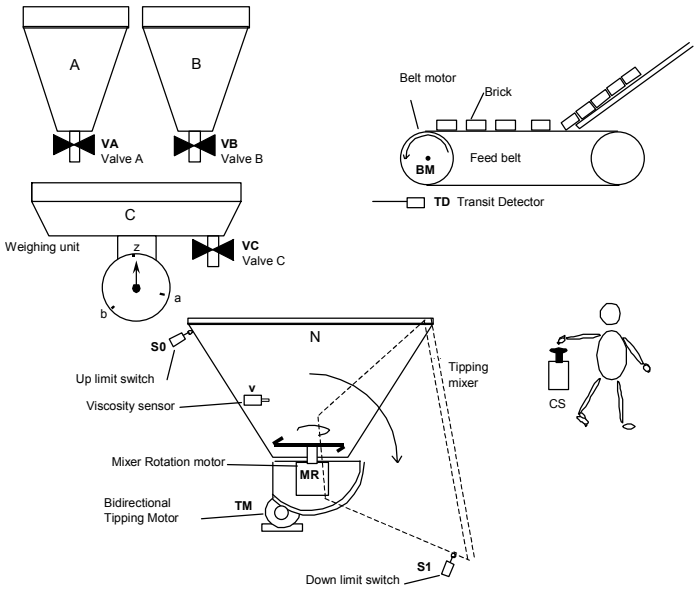


Fig. 7. Case study: an automatic weighing-mixing system

Table 1. Table of inputs & outputs

Input	Description
CS	*True* when a cycle starts
TD	*True* when a brick is detected
a	*True* when fluid weight A is reached
b	*True* when fluid weight A + B is reached
z	*True* when fluid is emptied
S0	*True* when Mixer is up
S1	*True* when Mixer is down
v	*True* when preset viscosity is reached

Output	Description
BM	turn on Belt Motor
MR	turn on Mixer Rotation Motor
TMp	Tipping Motor tips down
TMm	Tipping Motor tips up
VA	open Valve A
VB	open Valve B
VC	open Valve C

In each work cycle, two products are weighed and poured into a mixer. Meanwhile two bricks are transported into the mixer through a feed belt. The mixer mixes all the products and tips off the mixture after a preset viscosity is detected.

In total, 6 FSM models haven been made for the specifications. As examples, the two models for weighing and tipping processes are presented in Fig. 8, respectively.

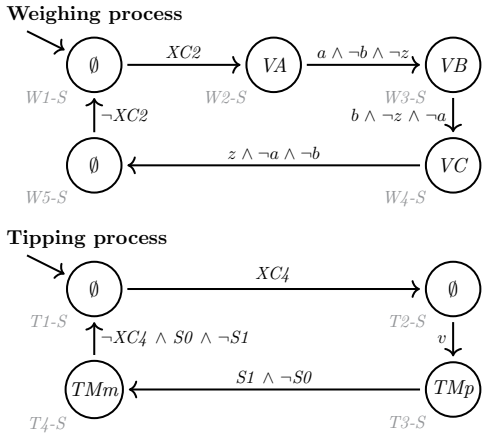


Fig. 8. Two out of six specification models

4.2 Complete test case generation

A complete test considers all evolutions originating from all locations and for all possible combinations of input signals.

Test cases are generated from the composed model. In this case study, the composed model is obtained by applying Teloco (Provost et al. (2011)) and contains 19 locations and 50 evolutions. Its structure is presented in Fig. 9.

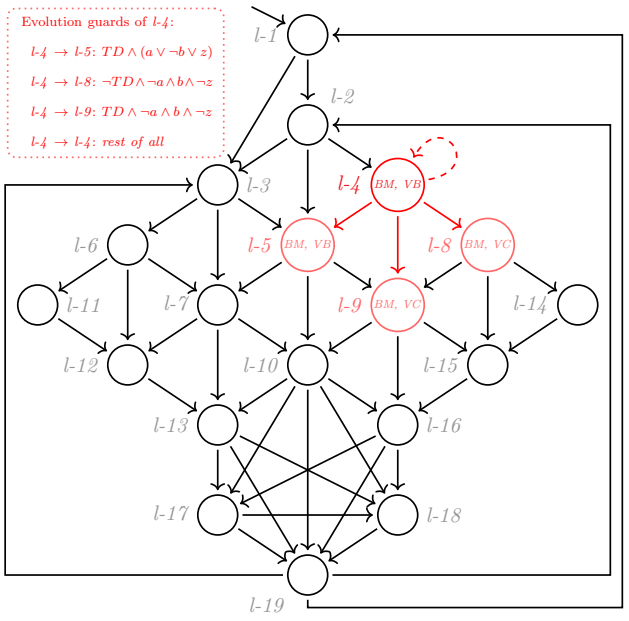


Fig. 9. The composed model of all specifications

In the following part, the location *l-4* and its evolutions (marked red in Fig. 9) are used to illustrate the differences between the existing and proposed methods for test case generation. The actions of locations *l-4*, *l-5*, *l-8* and *l-9*, and the evolution guards originating from *l-4* are presented in Fig. 9. More details on the composed model can be found in Ma and Provost (2015).

With 8 Boolean input signals, if the classic method is applied, then 256 ($= 2^8$) test cases are generated for each location. Thus, a complete test set of the whole system is made up of 4864 ($= 19 * 256$) test cases.

4.3 Test case reduction with plant models

Applying the proposed method, plant models are constructed to help reduce the test cases. For the case study, 5 plant models are made. Two of them, the weighing and tipping process, are presented in the paper.

With natural language, the following signal dependencies can be found for *l-4*:

Weighing process:

- The inputs z, a, and b in the weighing process can be *True* only when *CS* is activated.
- The input z is *True* before the actuator *VA* or *VB* is turned on.
- When *VA* or *VB* keeps turning on, a and b can become *True* one after the other.
- After *VC* is turned on, b, a, and z change their values conversely.

Tipping process:

- The inputs *S0* and *S1* in the tipping process can be *True* only when *CS* is activated.

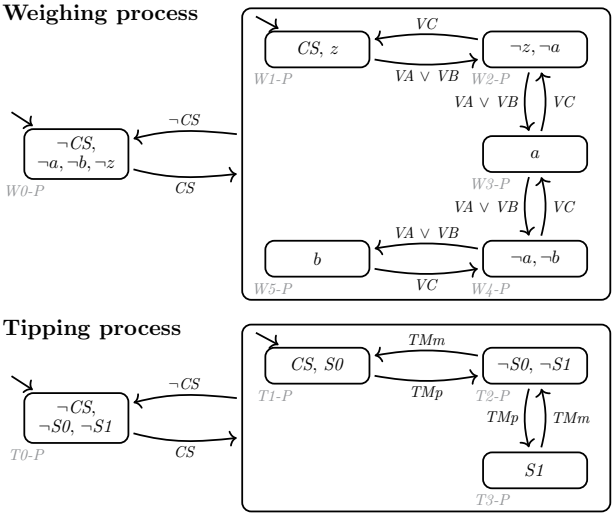


Fig. 10. Two out of five plant models [2]

The same models can be expressed with temporal logic (CTL as an example) as follows:

Weighing process:

- $AG(\neg CS \rightarrow (\neg z \wedge \neg a \wedge \neg b)) \wedge AG\neg(z \wedge a) \wedge AG\neg(a \wedge b) \wedge AG\neg(b \wedge z)$

Tipping process:

- $AG(\neg CS \rightarrow (\neg S0 \wedge \neg S1)) \wedge AG\neg(S0 \wedge S1)$

In Fig. 10, the signal relations are presented in the form of FSM.

[2] Plant models are used to describe discrete signal relations among sensors and actuators. The signal lasting effect between two discrete states is a continuous process and is therefore not considered.

When combining the plant and specification models, in the scope of *l-4*, the tipping process stays in its initial state, i.e. *S0*. Thus, *S1* is always *False* during the testing of *l-4*.

After composing all the 6 specification and 5 plant models, the number of nominal test cases for *l-4* is reduced from 256 to 9. The results are listed in Tab. 2.

Table 2. Test cases for evolutions from *l-4*

Inputs (CS, TD, a, b, z, S0, S1, v)	**Evolution under test**
1 0 1 0 0 1 0 0	*l-4* → *l-4*
1 0 0 0 0 1 0 0	*l-4* → *l-4*
1 0 0 0 1 1 0 0	*l-4* → *l-4*
1 1 0 0 0 1 0 0	*l-4* → *l-5*
1 1 0 0 1 1 0 0	*l-4* → *l-5*
1 1 1 0 0 1 0 0	*l-4* → *l-5*
1 0 0 1 0 1 0 0	*l-4* → *l-8*
1 1 0 1 0 1 0 0	*l-4* → *l-9*
0 0 0 0 0 0 0 0	*l-4* → *l-4*

For the whole system, the total number of test cases is reduced from 4864 to 158.

5. CONCLUSION

Complete conformance testing provides a best guarantee that a programmable controller will work properly according to its specifications in industrial automation systems. However, it is not widely applied owing to the huge number of test cases generated directly from specification models.

For large scale systems where only some subparts are critical, an efficient testing strategy is to consider only nominal behavior of a *system under test* with regard to physical constraints. This paper reaches this goal by utilizing features extracted from plant models. With the proposed method, plant models can be constructed with natural language, finite state machine or temporal logic formalisms. A smaller set of test cases which contains mainly nominal test cases can be obtained after composing specification and plant models. The result on a benchmark case study shows that the number of test cases is reduced significantly.

It is also worth mentioning that the framework does not require a detailed or full plant model. Any piece of plant knowledge can contribute to the test cases reduction. For future work, we plan to extend this method to faulty plant models where different types of faults can occur. A formal framework will be established to handle and sort the faults in testing and applied on a larger scale case study.

REFERENCES

Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., and Sa'Ar, Y. (2012). Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, 78(3), 911–938.

Campos, J.C., Machado, J., and Seabra, E. (2008). Property patterns for the formal verification of automated production systems. In *17th World Congress The International Federation of Automatic Control (IFAC)*, volume 17, 5107–5112.

Fantechi, A., Gnesi, S., Ristori, G., Carenini, M., Vanocchi, M., and Moreschini, P. (1994). Assisting requirement formalization by means of natural language translation. *Formal Methods in System Design*, 4(3), 243–263.

Frey, G. and Litz, L. (2000). Formal methods in PLC programming. *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, 4, 2431–2436.

Girault, A., Lee, B., and Lee, E.A. (1999). Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6), 742–760.

Gu, F., Harrison, W.S., Tilbury, D.M., and Yuan, C. (2007). Hardware-in-the-loop for manufacturing automation control: Current status and identified needs. *IEEE International Conference on Automation Science and Engineering*, 1105–1110.

IEC 60848 (2011). *GRAFCET specification language for sequential function charts*. International Electrotechnical Commission, 3. edition.

Jeon, J.H., Kim, J.Y., Kim, H.M., Kim, S.K., Cho, C., Kim, J.M., Ahn, J.B., and Nam, K.Y. (2010). Development of hardware in-the-loop simulation system for testing operation and control functions of microgrid. *IEEE Transactions on Power Electronics*, 25(12), 2919–2929.

Lee, D., Sabnani, K.K., Kristol, D.M., and Paul, S. (1996). Conformance testing of protocols specified as communicating finite state machines - A guided random walk based approach. *IEEE Transactions on Communications*, 44(5), 631–640.

Lee, W.C. and Drury, D. (2013). Development of a hardware-in-the-loop simulation system for testing cell balancing circuits. *IEEE Transactions on Power Electronics*, 28(12), 5949–5959.

Ma, C. and Provost, J. (2015). Design-to-test approach for black-box testing of programmable controllers. In *IEEE Int. Conf. on Automation Science and Engineering (CASE)*, 1018–1024.

Ma, C. and Provost, J. (2016). DTT-MAT: A software toolbox of design-to-test approach for testing programmable controllers. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, 878–884. Fort Worth, Texas, USA.

Machado, J.M., Denis, B., Lesage, J.J., Faure, J.M., and Ferreira Da Silva, J.C.L. (2006). Logic controllers dependability verification using a plant model. *IFAC Proceedings*, 39(17), 37–42.

Provost, J., Roussel, J.M., and Faure, J.M. (2011). Translating grafcet specifications into mealy machines for conformance test purposes. *Control Engineering Practice*, 19(9), 947–957.

Provost, J., Roussel, J.M., and Faure, J.M. (2014). Generation of single input change test sequences for conformance test of programmable logic controllers. *IEEE Trans.on Ind. Inform.*, 10, 1696–1704.

Ramadge, P.J. and Wonham, W.M. (1987). Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1), 206–230.

Rösch, S., Ulewicz, S., Provost, J., and Vogel-heuser, B. (2015). Review of model-based testing approaches in production automation and adjacent domains — Current challenges and research gaps. *Journal of Software Engineering and Applications*, 8(9), 499–519.