A model-based testing framework with reduced set of test cases for programmable controllers

Canlong Ma and Julien Provost

Abstract— **In testing of programmable controllers, manual selection of test cases is still the most common method in practice. This is however tailor-made, time consuming and error-prone. Traditional model-based methods can hardly handle industrial scale systems which usually possess a significant number of states, and signals of sensors and actuators. In this paper, we propose a model-based testing framework that utilizes simplified plant features to reduce the number of test cases, and at the same time also guarantees a full coverage of nominal behavior of system under test. The proposed framework has been illustrated on a case study.**

I. INTRODUCTION

Nowadays, evolving technologies in the field of industrial automation result in challenges of higher system complexity, more information exchange, more intelligence, more customization, and shorter life cycles [1], [2]. Thus, testing techniques with high reliability and efficiency have become an urgent demand.

In an industrial automation system, sensors and actuators are usually considered as *plants* while controllers are implemented according to *specifications*, i.e. formal descriptions of user requirements.

To test such controllers, manual selection of test cases is an old but still the most commonly used method. These manual methods are usually expert-based and have been proven useful in practice decades-long [3], [4], etc. Nevertheless, their disadvantages are obvious: individually customized, time consuming, and error-prone. The shortcomings have become big obstacles and hardly bearable for current developments.

Automatic testing techniques, especially model-based methods, have been investigated since long. In the scope of discrete event systems, a *test sequence* is an executable sequence of *test cases* passing through different states and transitions. The core matter of a test sequence is its length, which is determined by two factors: 1. the number of test cases; 2. the ordering and repetition of test cases. The second factor comes into being because in practice, a state can have several outgoing transitions, and some states have more transitions to be tested than others. Therefore, in a test sequence, some transition arcs need to be traversed several times.

Complete conformance testing (CCT) is a model-based technique that is highly advantageous for safety critical systems, since it considers all possible combinations of input signals from all states and therefore covers the whole behavior of a system under test. The limitation is that the number of test cases and subsequently the length of a test sequence grows exponentially with the number of inputs, which severely restricts its application to large-scale systems.

Efforts have been made for improvement by modifying the generation methods of test cases and test sequences. For instance, a design-to-test (DTT) approach for programmable controllers has been proposed in [5]. This permits to automatically reduce the testing overhead of a controller by inserting some control and observation points in the initial specification models. As a result, the length of a complete test sequence can be shortened while the number of test cases is kept unchanged. Yet, the test objective remains CCT, and thus, is more suitable for critical systems.

Other recent works aim to reach a high test coverage with a relatively small set of test cases. For example, [6] generates test cases based on the element identifier and function block-tree traversal; [7] uses coverage metrics to implement a symbolic execution engine. However, no plant behavior of system under test was considered in creating the coverage criteria. Thus, faulty behavior might be missed in these high but not full coverage testings.

In this paper, a testing framework is proposed that guarantees a full coverage of nominal behavior of a system and also offers the possibility to consider faulty behavior. The core idea is to involve not only specification models but also plant features in the test case generation. These plant features are extracted from simplified plant models and thus require a limited design effort. As a result, the number of generated test cases and the length of a test sequence could be significantly reduced, and therefore, large-scale systems can be tested more efficiently.

Compared to [5], the shortening of a test sequence with the proposed framework is obtained from a reduced set of test cases. Compared to [6] and [7], the obtained set of test cases is not reduced stochastically but is selected in a way that guarantees a full coverage of nominal behavior of a system.

The fundaments of the framework were presented in [8]. Compared to [8], the main contributions of this paper are:

- Formal realization of a test case generation framework
- Refinement of the algorithms
- Extension to testing of faulty behavior

The paper is structured as follows: Section II introduces the formalism of finite state machines used in specification and plant models. Sections III and IV present an overview of the testing process on programmable controllers, and detailed techniques and algorithms used in the generation of test

Canlong Ma and Julien Provost are with Assistant Professorship for Safe Embedded Systems, Technical University of Munich, Garching bei München, Germany `ma@ses.mw.tum.de`; `provost@ses.mw.tum.de`

cases, respectively. An industrial case study is illustrated in section V. Finally, the last section concludes this work.

II. Background

A. Communicating Moore machine with Boolean signals

Finite state machine (FSM) is a formal model widely used in model-based design and model-based testing. In this paper, specifications of a system are modeled as a set of Moore machines which can communicate with each other via internal variables, adapted from [9]. Practical control specifications using other formalisms such as IEC61131 [10] and GRAFCET [11] can be transformed into Moore machine, in order to apply the proposed method.

For the sake of brevity, Boolean signals are used as inputs and outputs in the illustration of the proposed method. However, it is possible to handle general digital signals with a few adaptations.

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

- S_S is a finite set of states[1]
- s_{init_S} is the initial state, $s_{init_S} \in S_S$
- I_S is a finite set of Boolean input signals
- C_S is a finite set of internal Boolean communicating variables
- O_S is a finite set of Boolean output signals
- $\delta_S : S_S \times 2^{I_S \cup C_S} \rightarrow S_S$ is the transition function that maps the current state and Boolean expression[2] to the next state
- $\lambda_S : S_S \rightarrow 2^{O_S \cup C_S}$ 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. A state s is drawn as a rectangle with rounded corners. A transition δ is represented by an arc with its guard, e.g. $\neg a \wedge b$ for the transition from s_1 to s_2. A state can either have an externally observable action[3], e.g. O_2 in s_2, or no observable action, e.g. $\emptyset$ in s_1. Additionally, a state can also be given an internal communicating variable, e.g. *XS6* in s_6, which can be used in Boolean transition guards. For example, when the state s_6 is activated, *XS6* is then assigned the value '1'. If s_2 is active at the same time, then the transition from s_2 to s_3 can be fired.

B. Composition of individual specification models

For complex applications, it is convenient to build several simple individual models and compose them synchronously, instead of constructing a large monolithic model directly.

A significant research effort has been done for the composition of FSM models. In this paper, the tool Teloco [11] has

[1]The subscript 'S' stands for 'Specification'.

[2]Boolean operators used in this paper: $\wedge$: AND; $\vee$: OR; $\veebar$: XOR; $\neg$: Negation.

[3]For readability reasons, only active outputs are presented, i.e. in s_2, O_2 implicitly means $O_2 \wedge \neg O_3 \wedge \neg O_4 \wedge \neg O_5$.

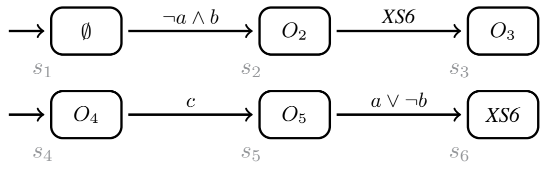

Fig. 1. A simple example of a Moore machine with Boolean signals

been used. It is worth mentioning that the obtained composed model contains only stable locations, i.e. locations where only a change in the input values can trigger a change of locations. This feature is called stability search.

The formalism of Moore machine is also used for a composed machine with following modifications and comments:

- L is the set of locations in a composed model. A location represents a combination of states from the individual models.
- $e_{(I_S,l)} : L \times 2^{I_S} \rightarrow L$ is the evolution function with stability search that maps the current location and a Boolean expression to the next location. A *transition* between locations is named an *evolution*.
- L_U (resp. L_D) is the set of immediately upstream (resp. downstream) locations of evolutions, $L_U, L_D \subset L$
- $E_{(I_S,L)} = \{e_{(I_S,l)}\}$ is the set of evolution guards, i.e. Boolean expressions on inputs and locations.

C. Description of nominal signal relations using FSM

In [8], three forms were introduced to model signal relations in plant models: natural language, temporal logic and finite state machine. Two core types of nominal signal relations using FSM are reminded below: *premise* and *mutual exclusion*. In practice, complex signal relations can be modeled using the two core types, especially when involving several signals. In this paper, the term *nominal behavior* refers to the behavior of a system without sensors and actuators faults. In practice, sensors and actuators can be faulty; the *faulty behavior* of a system is obtained after integration of their fault models to the *nominal behavior*; this will be introduced in Sec. IV.

1) Premise: As presented in Fig. 2, i_1 and i_2 are two Boolean signals. i_2 can only be *True*, when i_1 is *True*. i_1 is called a 'premise' of i_2.

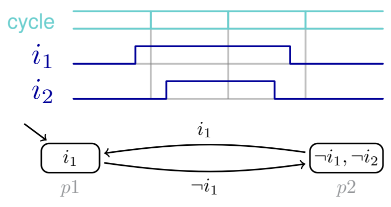

Fig. 2. Signal sketch and FSM model of premise relation[4]

According to this model, any test case where i_2 is *True* while i_1 is *False* can be excluded from the nominal behavior,

[4]Signals values can be freely assigned if they do not appear in the initial state, i.e. the output of p_1 can either be (i_1, i_2) or $(i_1, \neg i_2)$.

and is therefore removed from the set of nominal test cases. Thus, a single premise relation of two signals can reduce the total number of test cases by 25%.

2) Mutual exclusion: As presented in Fig. 3, at the same time only one of i_1 and i_2 can be *True*. The signals i_1 and i_2 are called 'mutually exclusive'.

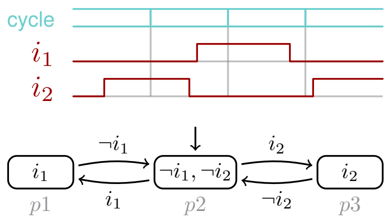

Fig. 3. Signal sketch and FSM model of mutual exclusion relation[5]

Similarly, the case that i_1 and i_2 are both *True* at the same time is not considered as a part of the nominal behavior. A single mutual exclusion relation of two signals can also reduce the total number of test cases by 25%.

III. Testing Objective and Process

The objective of conformance testing is to check whether the behavior of an implemented programmable controller conforms to the behavior of its specification model [12].

As presented in Fig. 4, a testing process consists of four steps:

- **Step 1:** Generate *test cases* and build a *test sequence*
- **Step 2:** Feed the input sequence to the programmable controller
- **Step 3:** Execute the implemented program on the controller
- **Step 4:** Compare the output sequences

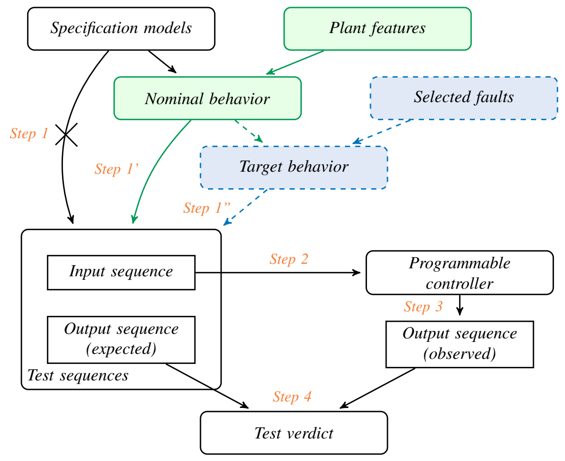

Fig. 4. Framework of involving plant features in testing of programmable controllers

The focus of this paper lies in the first step: generation of *test cases* and construction of a test sequence, and more specifically in the test case generation.

[5]The value of a signal remains unchanged until it is deliberately modified, i.e. the explicit output of p_1 is $(i_1, \neg i_2)$.

A classic process of this step has been presented in [11]: firstly, all individual specification models are composed to obtain a flattened Moore machine model; then, an equivalent Mealy machine model is built from the composed Moore machine model by explicitly representing all Boolean conditions of an evolution by a set of minterms over the Boolean input set; the last stage is to construct a test sequence, as introduced in Sec. I. Here, this is realized by solving the Transition Tour problem of the set of minterms from all states and all input values.

The method proposed in this paper follows this process except that, instead of considering the whole set of minterms from all locations, only the sets of minterms that correspond to the nominal behavior are selected to generate test sequences. The algorithms are presented in Sec. IV.

Formally, the following definitions are used in the sequel of this paper:

- Each $e_{(I,l)} \in E_{(I,L)}$ can be expressed by a set of minterms M_e. The Boolean condition is *True* for all and only $m_e \in M_e$.
- $i(m_e)$ represents the values of inputs of a minterm, i.e. *True* as 1 and *False* as 0.
- $o(l_u, m_e)$ represents the values of outputs of the upstream location of the evolution for the minterm m_e[6], i.e. *True* as 1 and *False* as 0.

For example, for $e_{(I,l)} = a \wedge \neg b$ with $I = \{a, b, c\}$, the corresponding M_e is $\{a \wedge \neg b \wedge c, a \wedge \neg b \wedge \neg c\}$, and the $i(m_e)$ is accordingly $\{(1,0,1),(1,0,0)\}$.

A crucial advantage of this framework is that it does not require a very detailed or full plant model, but only fragments of knowledge from plant models. Of course, if several plant models can be built, the number of nominal minterms (i.e. the number of test cases), and subsequently the length of a test sequence can be significantly reduced.

Additionally, users can also insert a set of faults in order to extend the set of minterms of interest to faulty behaviors.

IV. Test Case Generation with Utilization of Plant Models

In this paper, plant features are sorted into two levels.

A. First level: Signal relations among sensors

The Moore machine syntax introduced in Sec. II-A is also used to model plant features.[7]

On the first level, signal relations are built among sensor signals, which are used as inputs in specification models. Thus, the inputs and outputs of a plant model are both inputs from specification models. Formally, following definitions apply on the plant models of level 1:

- $I_{P_{L1}} = I_S$
- $O_{P_{L1}} = I_S$

The algorithm to generate minterms of interest is presented in Alg. 1. P_{L1} and M_e are the inputs and represent the set of

[6]Similar functions $i(s)$ and $o(s)$ applied on a state are used in the algorithms in Sec. IV.

[7]The subscript 'P_{L1}' is used to denote plant model level 1 (resp. 'P_{L2}' for level 2).

plant models and the set of minterms of an evolution guard $e_{(I,l)}$, respectively. M_{eT} is the output and represents the set of minterms of interest of an evolution.

In short, only minterms which are consistent with plant features among sensors will be added to the nominal set, which will be used to generate the set of test cases later on.

Algorithm 1: Generation of nominal minterms of an evolution with consideration of signal relations level 1

```
   Input: $P_{L1}, M_e$
   Output: $M_{eT}$
1  Initialization: $M_{eT} := \emptyset$;
2  begin
3      foreach $m_e \in M_e$ do
4          $Flag_{Remove} := False$;
5          foreach $p \in P_{L1}$ do
6              $I_\Sigma := zeros(length(I_{P_{L1}}))$;
7              foreach $s \in S_p$ do
8                  $I_C := i(m_e) \veebar i(s)$ ;
                   /* check consistency of minterm input
                      values to those allowed by plant   */
9                  $I_\Sigma := I_\Sigma + I_C$;
10             if $I_\Sigma \neq zeros(length(I_{P_{L1}}))$
               /* a minterm voilates at least one plant  */
11             then
12                 $Flag_{Remove} := True$ ;
13                 $break$;
14         if $Flag_{Remove} = False$
           /* a minterm is consistent with all plants   */
15         then
16             $M_{eT} := M_{eT} \cup \{m_e\}$ ;
               /* add a minterm to the set of interest   */
```

B. Second level: Signal relations among sensors and actuators

On this level, plant models are built using actuator signals to restrict sensor signals. Thus, the inputs and outputs of a plant model are outputs and inputs from specification models, respectively. Similarly, following definitions apply on the plant models of level 2:

- $I_{P_{L2}} = O_S$
- $O_{P_{L2}} = I_S$

In Alg. 2, inputs and outputs are defined in the same way as Alg. 1. Similarly, only minterms which are consistent with plant features among sensor and actuators are added to the nominal set of minterms to be tested.

C. Test case generation with fault injection

Fault injection is a class of testing techniques which involves faulty behavior supplementary to nominal behavior testing. The faults to be tested are usually selected based on expert knowledge and practical experience. For example, some components might be more error-prone in some environment, and some sensors might have physical interference with other sensors or actuators. More fault injection

Algorithm 2: Generation of nominal minterms of an evolution with consideration of signal relations level 2

```
   Input: $P_{L2}, M_e$
   Output: $M_{eT}$
1  Initialization: $M_{eT} := \emptyset$ ;
2  begin
3      foreach $m_e \in M_e$ do
4          $Flag_{Remove} := False$;
5          foreach $p \in P_{L2}$ do
6              $I_\Sigma := zeros(length(I_{P_{L2}}))$;
7              foreach $s \in S_p$ do
8                  $I_C := i(m_e) \veebar i(s)$ ;
9                  $O_C := o(l_u, m_e) \veebar o(s)$ ;
                   /* check consistency of output values
                      related to a minterm, to those
                      allowed by plant                    */
10                 $I_\Sigma := I_\Sigma + (I_C \veebar O_C)$;
                   /* check consistency between inputs and
                      outputs                             */
11             if $I_\Sigma \neq zeros(length(I_{P_{L2}}))$
12             then
13                 $Flag_{Remove} := True$ ;
14                 $break$;
15         if $Flag_{Remove} = False$
16         then
17             $M_{eT} := M_{eT} \cup \{m_e\}$ ;
```

knowledge and techniques in the field of testing can be found in [13].

In this paper, fault injection can be realized conveniently by modifying plant models. By definition, fault models are in conflict with the plant feature models described in Sec. IV-A and Sec. IV-B. Thus, test cases representing faulty behavior of signals are added to the set of test cases when the corresponding nominal signal relations are removed. When sensors and actuators faulty behaviors have to be considered during testing, this implies testing more test cases outside of the nominal behavior.

V. Case study : A welding and material handling cell

A large-scale industrial case study (Fig. 5) adapted from [14] and [15] is used in this paper to illustrate this approach.

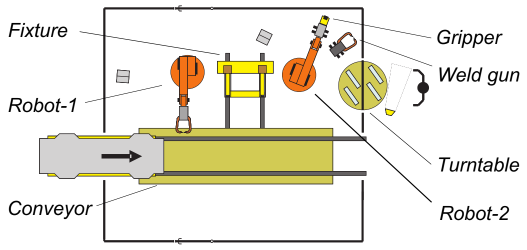

Fig. 5. Case study: a welding and material handling cell

A. Description of the system

As presented in Fig. 5, a manufacturing cell contains five machines: *Robot-1*, *Robot-2*, *Fixture*, *Turntable* and *Conveyor*. This system is modeled with 18 input and 17 output signals. As an example, the signals related to *Robot-2* are listed in Tab. I. Note: for *Ji-R2-done* and *Weld-Ji-R2*, it applies $i \in \{1, 2, 3\}$.

TABLE I
TABLE OF INPUTS & OUTPUTS RELATED TO *Robot-2*

Input	*Ji-R2-done*	*Away-R2*	*Present-R2*
	Weldgun-R2	*Gripper-R2*	*Plate-in-F*
Output	*Weld-Ji-R2*	*Move-away-R2*	*Move-back-R2*
	Pick-place-R2	*G2W-R2*	*W2G-R2*

At the beginning of a work cycle, *Conveyor* delivers a car body into the cell. *Robot-1* welds the parts that are previously loaded, which is named *Job-1*. Meanwhile, *Robot-2* picks plates from *Turntable* and places them in *Fixture*. *Turntable* turns when two plates have been taken. After that, *Robot-2* changes its tool to weld the plates held by *Fixture* to the car body together with *Robot-1*. This is named *Job-2*. Then, *Fixture* moves away from its workstation, to enable *Robot-1* and *Robot-2* weld the parts which were blocked. This is *Job-3*. When the weld jobs are done, the robots move away, so that *Conveyor* can take the car body out of the cell. In the end of a cycle, all the machines move back, and *Robot-2* changes back its tool to gripper.

Six Moore machines have been modeled for the specifications. For the sake of brevity, a model for *Robot-2* is selected as an illustrative example in this paper (Fig. 6).

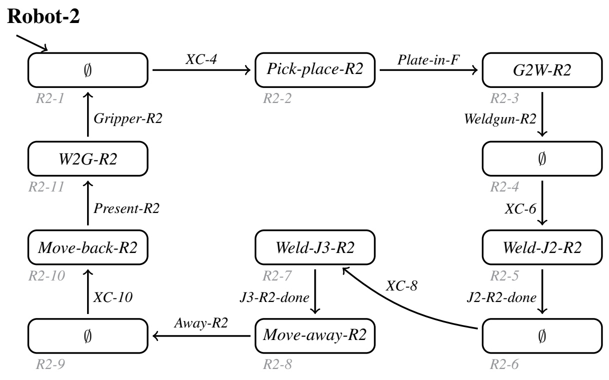

Fig. 6. A Moore machine model for *Robot-2*

B. Complete test case generation

Applying Teloco [11], the composed model of the six individual models contains 96 locations and 3,160 evolutions. Since the system has 18 inputs, a complete set of test cases would contain $96 * 2^{18} =$ 25,165,824 elements. Therefore, the length of a complete test sequence would be even larger.

The next subsections will present the test case generation processes with plant models level 1, with plant models level 2, and with fault injection, respectively. A summary of the results is presented in Tab. II.

C. Plant features level 1

The first level plant models of *Robot-2* are presented in Fig. 7 as an example.

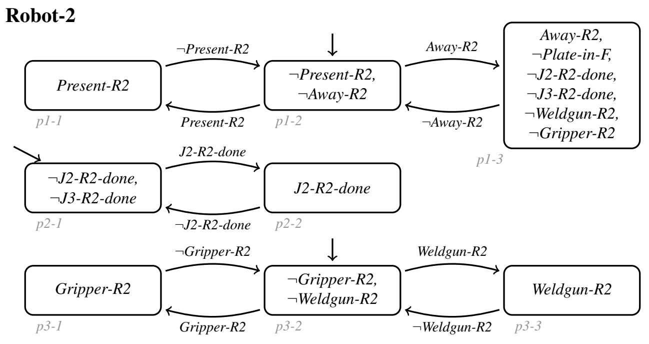

Fig. 7. Plant models for the nominal behavior of the *Robot-2*

In *p1*, i.e. the first plant model, *Present-R2* and *Away-R2* are mutually exclusive, since a robot cannot be ‘at’ and ‘away from’ its workstation simultaneously. Besides, some other signals should not be *True* when *Robot-2* is away from its workstation (state *p1-3*). In *p2*, *J2-R2-done* is a premise of *J3-R2-done*, because the plate is not fixed to the car body until *Job-2* is finished. If *Fixture* releases, the plate will fall and *Job-3* cannot be operated at all. In *p3*, *Gripper-R2* and *Weldgun-R2* are mutually exclusive, because both tools cannot be mounted to *Robot-2* simultaneously.

It is worth noting that, signals relations also exist among sensors on different machines, which are not presented in Fig. 7, but are easily conceivable. For example, *J2-R2-done* is also a premise of *J3-R1-done*, because *Job-3* of *Robot-1* can only be started when *Robot-2* also finished *Job-2*.

Applying all these signal relations among sensors, the number of test cases is reduced to 552,960.

D. Plant features level 2

To further shrink the number of test cases, signal relations among sensor and actuators are also extracted. The models for *Robot-2* are presented in Fig. 8.

In *ps1*, when *Robot-2* does the action *Move-away-R2*, the sensor signal *Present-R2* should not be *True*. The same relation exists between *Move-back-R2* and *Away-R2*. In *ps2*, in a nominal situation, the signal *Ji-R2-done* cannot be *True* before the corresponding action *Weld-Ji-R2* has been taken. Once all welding jobs have been done and the robots have moved away, *Ji-R2-done* is reset to *False*. In *ps3*, there could be no plate detected before *Robot-2* picks and places one to *Fixture*. After *Fixture* has moved away, the signal *Plate-in-F* is reset to *False*. In *ps4*, it is not difficult to find that only after the tool change action *W2G-R2* is taken, *Gripper-R2* can be detected as the current tool of *Robot-2*. The only exception is that *Robot-2* holds *Gripper-R2* at the very beginning as specified in the initial state. The same relation exists between *G2W-R2* and *Weldgun-R2*.

Combining the signal relations among sensors and actuators, the number of test cases is further reduced to 377,562.

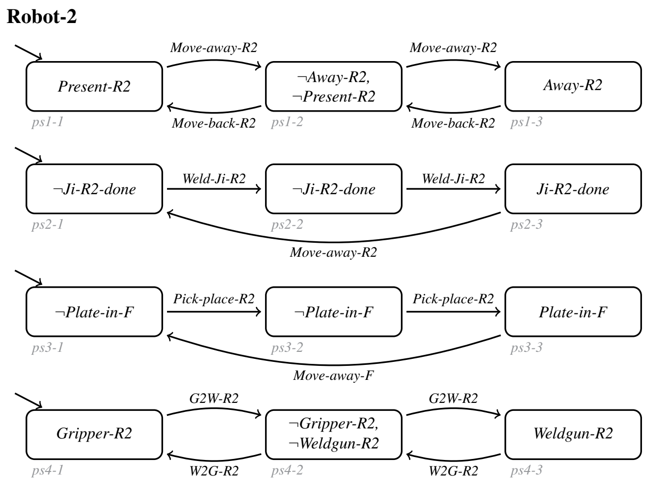

Fig. 8. Interaction between plant and specification models of *Robot-2*

E. Test cases with faults

Faults can occur in various sensors and actuators. As an example, let us suppose a sensor which recognizes whether a weld gun or gripper is mounted to *Robot-2* becomes faulty after operation for a long time. Moreover, the sensor signals *Weldgun-R2* and *Gripper-R2* are of great importance to the system, since an incorrect recognition of a weld guns and a gripper can lead to heavy damage to both a car body and the robot.

In order to involve the above-discussed considerations in the test cases, faults are injected by removing the third model in Fig. 7 and the fourth model in Fig. 8.

After updating the plant models, a set of 528,912 test cases is obtained.

F. Comparison of results

The results of different test case generation processes are presented in Tab. II. In summary, with the proposed framework a significantly smaller set of test cases is obtained compared to the traditional CCT.

TABLE II
COMPARISON OF DIFFERENT TEST CASE GENERATION RESULTS

Generation technique	Number of test cases	Compared to previous result	Compared to CCT
CCT	25,165,824	-	-
after using level 1	552,960	-97.8%	-97.8%
after using level 2	377,562	-31.7%	-98.5%
after fault injection	528,912	$+40.1\%$	-97.9%

VI. CONCLUSIONS

This paper presented a model-based testing framework to reduce the number of test cases by utilizing signal features extracted from simplified plant models. Plant features are modeled as finite state machines on two levels: signal relations among sensors, and signal relations among sensors and actuators.

Compared to traditional complete testing where test cases are generated directly from specification models only, the proposed framework obtains a significantly reduced set of test cases while still fulfilling a full coverage of the whole nominal behavior of a system under test. In addition, the proposed framework provides users an option to insert a selected set of faults into the target behavior to be tested.

It is worth mentioning that the framework does not require a detailed or full plant model. Any fragment of plant knowledge can contribute to the test cases reduction.

REFERENCES

[1] R. Schmidt, M. Möhring, R.-C. Härting, and C. Reichstein, “Industry 4.0 - Potentials for creating smart products: empirical research results,” in *International Conference on Business Information Systems*. Springer International Publishing, 2015, pp. 16–27.

[2] P. Leitão, A. W. Colombo, and S. Karnouskos, “Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges,” *Computers in Industry*, vol. 81, pp. 11–25, 2016.

[3] J. Mcgregor, “Testing a software product line,” Carnegie Mellon University, Tech. Rep., 2001.

[4] E. Jee, D. Shin, S. Cha, J.-s. Lee, and D.-h. Bae, “Automated test case generation for FBD programs implementing reactor protection system software,” *Software Testing, Verification and Reliability*, vol. 24, no. 8, pp. 608–628, 2014.

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

[6] P. Mani and M. Prasanna, “Automatic Test Case Generation for Programmable Logic Controller using Function Block Diagram,” in *International Conference on Information Communication and Embedded Systems (ICICES)*, 2016, pp. 1–4.

[7] D. Bohlender, H. Simon, N. Friedrich, S. Kowalewski, and S. Hauck-Stattelmann, “Concolic test generation for PLC programs using coverage metrics,” in *Discrete Event Systems (WODES), 2016 13th International Workshop on. IEEE.*, 2016, pp. 432–437.

[8] C. Ma and J. Provost, “Using plant model features in generation of test cases for programmable controllers,” in *20th World Congress The International Federation of Automatic Control*, 2017, p. to be publised.

[9] D. Lee, K. K. Sabnani, D. M. Kristol, and S. Paul, “Conformance testing of protocols specified as communicating finite state machines - A guided random walk based approach,” *IEEE Transactions on Communications*, vol. 44, no. 5, pp. 631–640, 1996.

[10] S. Lampérière-Couffin, O. Rossi, J. Roussel, and J. Lesage, “Formal validation of PLC programs: a survey,” in *European Control Conference (ECC)*, 1999, pp. 2170–2175.

[11] J. Provost, J. M. Roussel, and J. M. Faure, “Translating Grafcet specifications into Mealy machines for conformance test purposes,” *Control Engineering Practice*, vol. 19, no. 9, pp. 947–957, 2011.

[12] J. Provost, J.-M. Roussel, and J.-M. Faure, “Test sequence construction from SFC specification,” in *2nd IFAC Workshop on Dependable Control of Discrete Systems (DCDS’09)*, Bari, Italy, 2009, pp. 299–304.

[13] S. Rösch and B. Vogel-Heuser, “A Light-Weight Fault Injection Approach to Test Automated Production System PLC Software in Industrial Practice,” *Control Engineering Practice*, vol. 58, pp. 12–23, 2017.

[14] K. Andersson, B. Lennartson, and M. Fabian, “Restarting manufacturing systems; Restart states and restartability,” *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 3, pp. 486–499, 2010.

[15] C. Ma and J. Provost, “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)*, Fort Worth, Texas, USA, 2016, pp. 878–884.