SATE: Model-Based Testing with Design-to-Test and Plant Features

Canlong Ma, Claudius Jordan, Julien Provost

Technical University of Munich, Munich, Germany
(e-mail: {canlong.ma, claudius.jordan, julien.provost}@tum.de).

Abstract: In this paper we present SATE, a tool aiming at increasing test efficiency of model-based testing of DES using two approaches: design-to-test and plant features. First, the design-to-test approach automatically modifies the design while maintaining the original system behavior to overcome controllability, observability and SIC-testability issues. Secondly, testing with plant features reduces the number of test cases taking into account restrictions on the input space of programmable logic controllers caused by the plant that is to be controlled.

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

1. INTRODUCTION

It is commonplace that industrial automation systems grow larger, and thus the programmable logic controller code grows in terms of complexity. This results in the demand for tools that enable testing such controllers efficiently. Many model-based techniques exist in literature to generate test cases for black-box testing; one of those is complete conformance testing (CCT). In order to show complete conformance of an implementation to its specification by means of testing, all possible combinations of inputs need to be evaluated for all states. Conformance testing of a programmable controller consists of three phases: test generation, test execution, and result verdict.

Here, we focus on the test generation phase in order to reduce the overall length of a test sequence, which is an artifact of this phase. It is a challenging endeavor to create a complete set of test cases for large scale systems as the number of test cases grows exponentially with the number of states and inputs. In addition to the before-mentioned state space explosion, in many cases CCT suffers from single-input-change-testability (SIC-testability) issues discussed in Provost et al. (2014).

During test sequences execution, the actual test case evaluation is performed after reaching a certain source state. Thus, additional computations have to be made to actually reach this desired state to perform the considered test step. Moreover, if the system evolution is not observable or the system state after the test step is not distinguishable from others, it has to be identified with state identification techniques, which demand further effort (Lee et al. (1996a)). These two actions, the homing and identification sequences, constitute a testing overhead, which can be remarkably big for larger systems. Consequently, reducing this testing overhead is an aim for our design-to-test (DTT) approach (Ma and Provost (2016)). To achieve this, the system design is modified automatically, introducing some design overhead with the goal of reducing the testing overhead discussed beforehand.

The second methodology addresses the reduction of the number of generated test cases in terms of reducing the controller input space in each system state, taking into account which outputs the physical plant can actually produce. The underlying hypothesis is that in the closed-loop, some inputs for the controller will never occur under nominal system behavior and thus can explicitly be neglected during testing.

Even limited knowledge about the system, i.e. a single plant feature, already leads to a reduction of test cases in comparison to the CCT approach. There are some common relations between inputs that can easily be identified such as *mutual exclusion* and *premise*. Similarly, relations between outputs and inputs can be found. Those features are presented and discussed in more detail in Ma and Provost (2017), where also templates are provided.

Other approaches to generate test cases automatically from models can be found in the literature. For example Enoiu et al. (2013) and Mani and Prasanna (2016) use a model checker to generate test suites based on Function Block Diagrams. In Bohlender et al. (2016) high coverage is realized more efficiently by symbolic execution. In contrast to the before-mentioned approaches, we consider full coverage taking into account the behavior of the system under consideration. Kormann and Vogel-Heuser (2011) create a reduced set of meaningful test cases to be executed in a simulated environment for hardware based component faults, whereas we focus on nominal system behavior.

In this paper, we present our tool, Stable Automaton-based TEsting (SATE), that implements the two approaches that aim for more effective and shorter test sequences. In the next section, necessary background on communicating Moore machines and their synchronous composition are recalled. In Sec. 3 the framework and actual implementation are discussed accompanied by a case study illustrating the effectiveness of the presented tool in Sec. 4. This work is concluded with some remarks on potential future work in the last section.

2. BACKGROUND

2.1 Communicating Moore machine with Boolean signals

In this paper, system specifications are modeled as communicating Moore finite state machines, adapted from Lee et al. (1996b).

Due to simplicity 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, in contrast to event based models, where only one event can occur at a time, signal based models allow multiple changes of input values at once.

A communicating Moore machine extended with Boolean signals is defined by an 8-tuple $(L, l_{init}, I, C, O, G_\delta, \delta, \lambda)$ [1], where:

- L is a finite set of locations.
- l_{init} is the initial location, $l_{init} \in L$.
- I is a finite set of Boolean input signals.
- C is a finite set of internal Boolean communicating variables that are related to locations, a communicating variable is denoted as $X(l)$.
- O is a finite set of Boolean output signals.
- $G_\delta := expr(I, C)$ is a finite set of transition guards, which are Boolean expressions built up by input signals and communicating variables.
- $\delta : L \times G_\delta \to L$ is the transition function that maps the current location and transition guard to the next location; a transition is fired when its source location is active and its guard is evaluated as '1' (i.e. *True*).
- $\lambda : L \to 2^{O \cup C}$ is the output function that maps the locations to their corresponding output signals and communicating variables.

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

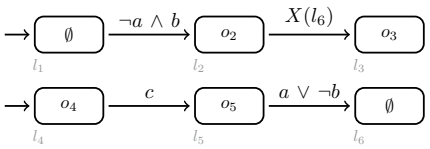

Fig. 1. A simple Moore machine example with Boolean signals

A location l is drawn as a rounded rectangle. A location can either have an externally observable action [2], e.g. o_2 in l_2; or no observable action, e.g. $\emptyset$ in l_1.

A transition δ is represented by a directed edge with its guard, e.g. $\neg a \wedge b$ for the transition from l_1 to l_2. The use of an internal communicating variable in transition guards is not complicated. For example, when the location l_6 is activated, $X(l_6)$ is then assigned the value '1'. If l_2 is active at that time, the transition from l_2 to l_3 can be fired.

[1] The subscript 'S' will be used to stand for *Specification*, the subscript 'P' for *Plant*: e.g. L_S and L_P mean the set of locations for specification and plant models

[2] For readability reasons, only active outputs are presented, i.e. in l_2, o_2 implicitly means $o_2 \wedge \neg o_3 \wedge \neg o_4 \wedge \neg o_5$.

2.2 Synchronous composition of individual models

Thanks to the use of internal communicating variables, interactions among individual parts of a system can be modeled conveniently. In order to validate the global behavior, individual models are first composed synchronously.

SATE's composition is based on the algorithms used by the tool 'Teloco' (Provost et al. (2011)), and the formalism introduced in Sec. 2.1 is extended with the following modifications:

- S is the set of states in a composed model. A state represents a combination of locations from the individual models.
- $G_e := expr(I)$ is a finite set of evolution guards [3], which are Boolean expressions built up by input signals.
- $e : S \times G_e \to S$ is the evolution function with stability search that maps the current state and evolution guard to the next state. A *transition* between states is named an *evolution*.

It is worth reminded that during the composition, a situation is stable if no transition in any of the Moore machines can be fired without changing the values of input signals; otherwise, it is transient. The stability search semantics implies that the firing of transitions continues until a stable situation is reached. The composed model contains only stable states, where only a change in the input values can trigger an evolution to another state. Therefore, the composed model is called *Stable Composed Automaton* (SCA) in this paper.

3. TEST CASE GENERATION METHODS

In this paper, the testing objective is to check whether an implemented programmable controller, seen as a black-box with inputs and outputs, behaves correctly with respect to its specifications. The execution of a testing process consists of three steps: feeding the input sequence to the controller, executing the program, comparing the observed output sequence to the expected one generated from specifications.

This paper focuses on the generation of test cases. As presented in Fig. 2, the process of *complete conformance testing* method is depicted in the center, while our *design to test* approach and methodology for *testing with plant features* are presented on the left and the right side, respectively.

3.1 Complete conformance testing

The basic complete conformance testing is structured and performed as follows. First, individual Moore machine models are modeled and composed into one global SCA. Afterwards, an equivalent Mealy machine is derived from the SCA, which explicitly represents all Boolean conditions of evolutions by a set of minterms [4] over the Boolean input

[3] A specific evolution guard is noted as $g_{e(I_S,s)}$, with regard to its source state and involved inputs

[4] A minterm is a basic element of an explicitly presented guard, e.g. if $g_{e(I,l)} = a \wedge \neg b$ and $I_S = \{a, b, c\}$, the corresponding minterms are $a \wedge \neg b \wedge c$ and $a \wedge \neg b \wedge \neg c$.

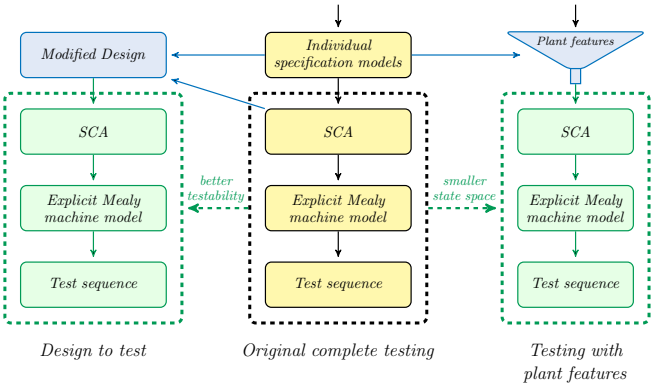

Fig. 2. Framework of complete testing, design to test, and testing with plant features methods

set. As a last step, a test sequence is generated by solving the Transition Tour problem of the set of minterms from all states and all input values.

3.2 Design to test approach

Several issues have been identified in the practice of complete testing regarding controllability, observability and SIC-testability. The design-to-test (DTT) approach aims to solve those issues by paying a limited effort on the modification of design, while keeping the nominal behavior during normal execution unchanged (Ma and Provost (2016)). SATE incorporates the algorithms of the 'DTT-MAT' toolbox.

A good design, which fulfills all functional requirements, is not always a good design with respect to testing. Two abstract Moore machine models (before and after the modification by DTT approach) are presented as examples in Fig. 3.

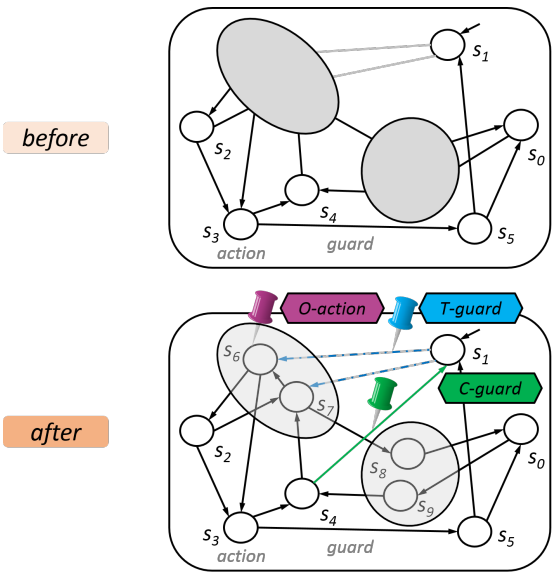

Fig. 3. Core idea of DTT approach: adding T-guards, O-actions and C-guards to modify the initial specification models

In the conformance testing of programmable controllers, controllability is a measure of whether and how fast the implementation can be brought from an (arbitrary) active state to another desired state. In Fig. 3, on the initial model, the shortest path from s_4 to s_1 is relatively long (with regard to the number of states in the example system), which represents a relatively bad controllability. The DTT approach solves this issue by adding a minimum number of extra controllable transitions, named as C-guard transitions, between some of the states (drawn with green color in the example), which can be used as shortcut transitions when set to *True*.

Observability concerns whether and how fast a state can be distinguished from other states. Apparently, in Fig. 3, the states in the two big gray circles cannot be distinguished directly by observation of system outputs in that moment, since they have the same outputs. The DTT approach solves this issue by adding extra observable actions, named as O-actions, to such states which suffer from the presented lack of observability.

SIC-testability issues occur in testing of controllers with cyclic execution mode, when several input signals are expected to change their values at the same time. Physically, multiple input changes (MIC) do not necessarily occur at the same time. Consequently, the changes might be read by the controller in different cycles. Then, the actual behavior may differ from the behavior under consideration that should be tested. Obviously, this would not be an issue in the case that the test sequence only contained single input changes between two successive steps, which is however hardly achievable in practical systems (Guignard and Faure (2014)). The DTT approach solves this issue by adding T-guards to the initial guards of transitions suffering from SIC-testability issues. All T-guards will be set to *False*, when multiple inputs should change at once, stopping the system at its current state such that there is enough time to stabilize the MIC. This enables to proceed as usual and make sure that the desired transition can be taken.

In summary, applying the design-to-test approach, the specification models are automatically analyzed by the tool; then, based on the need, a minimum number of C-guards, O-actions, and T-guards are automatically calculated and added to the models, so that the specification models fulfill the requirements of full SICtestability, full observability and better controllability.

3.3 Testing with plant features

With the DTT approach, complete testing can be done more effectively. However, for large scale systems, complete testing cannot always be achieved, because the number of test cases grows very rapidly to the complexity of a system, i.e. exponentially to the number of inputs and linearly to the number of states.

Therefore, a test generation method using plant features has been proposed for more efficient testing (Ma and Provost (2017)). The algorithms in SATE are based on the method in Ma and Provost (2017) but with improvements, i.e., applying plant features earlier in the generation of SCA rather than after the generation of explicit Mealy machine, which further reduces the state space of test generation. The concepts of plant and specification are

high	*False*	*False*	*True*	*True*
low	*False*	*True*	*False*	*True*

Table 1. Truth table for input combinations for the tank example. The non-nominal combinations (due to plant features) are marked in red.

nontrivial in automation systems. As presented in Fig. 4, sensors, actuators and all other physical elements (except the controller) are considered as plant. A controller is designed to control the plant, and implemented according to a set of specifications.

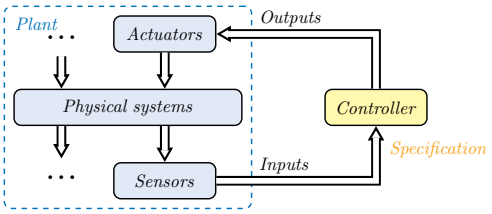

Fig. 4. Closed loop of plant and controller

By nature, the behavior of the plant is strongly influenced by the controller. However, as presented in Fig. 4, in a closed-loop system, the plant also restricts the reachable state space of the controller in normal operation. This leads to the idea of considering the closed-loop behavior also during test execution.

This hypothesis can be used to reduce the number of test cases by removing the cases that cannot occur in nominal behavior of a system, resulting in a reduced set of *meaningful* test cases. The undesired or unexpected behavior will not be tested during early test phases (or not at all if complete testing is not feasible). Given a tank with two level sensors: when the sensor indicating *Level High Reached* gives the value *True*, the sensor indicating *Level Low Reached* should normally not give the value *False*. A second example is a conveyor belt: if it does not run, the sensors for detecting the position of a workpiece should not change their value, since no workpiece has been moved and the sensors should not be triggered in a nominal situation. Such relations can be displayed in a truth table, indicating the possibility to reduce the number of input combinations as displayed in Tab. 1, where the combination of high and not low. Apparently, one out of four combinations can be neglected, leading to a reduction of 25% of test cases.

The aim of specifying relations between inputs as well as outputs and inputs is to reduce the set of states for which complete testing is performed. Effectively, this reduces the number of states and the input vectors. This results in fewer combinations of inputs that need to be imposed on the controller in open-loop testing. During composition, fewer states and evolutions need to be considered, which reduces the computational complexity.

4. CASE STUDY

In this paper, a logistics system is used as an illustrative example. The three subsystems are taken and adapted from the didactic platform presented in Jordan et al. (2017). The modules of interest in our case study are displayed in Fig. 5.

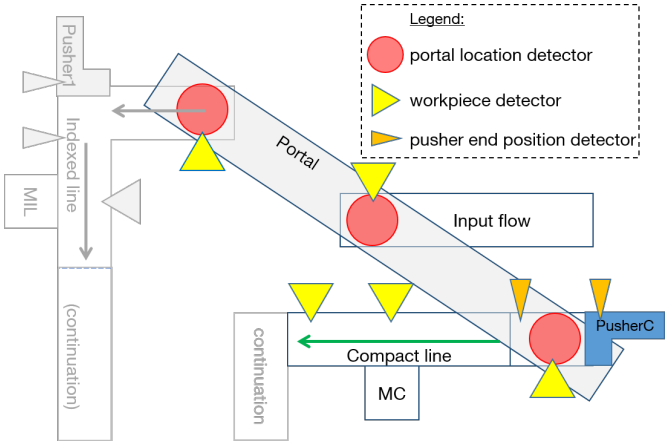

Fig. 5. Part of a logistics system containing a portal and two subsequent lines *(top view)*

4.1 System description

The *portal* transports workpieces from the *input buffer* to either the *compact line* or an indexed line. In this paper, the specification and plant behavior of the portal and compact line are analyzed and presented. The compact line contains a vertical buffer with a pusher, a conveyor belt and one machine station. Several location sensors are used to sense the position of the workpiece (yellow triangles), the pusher (orange longish triangles) and the portal (red circles).

Five FSM models have been used for the specification models of the system under consideration. In Fig. 6, three specification models for the portal and compact line are given as examples. [5] It is displayed, that the portal can move horizontally between three positions *In*, *IL* and *CL*. Only in those positions it can move up and down. Finally, only in the down end position it can activate the electromagnetic gripper to lift a workpiece or deactivate it (ungrip) in order to release a workpiece, respectively. On the compact line, a workpiece is brought to a machine via the belt; after the machining the workpiece is delivered to the output.

In total, 15 inputs and 9 outputs are considered, as listed in Tab. 2.

4.2 Applying DTT approach

For the sake of comparison, a monolithic composition of the system is performed resulting in an SCA with 221 states and 8524 evolutions.

After checking with DTT approach, all the 221 states contain non-SIC-testable parts; after adding T-guards to 8 transitions into the individual models, the system becomes fully SIC-testable.

In the SCA, 183 states suffer from observability issue. 5 O-actions are added to a set of locations in individual models, so that all states are directly distinguishable from each other.

[5] Initial specification models are drawn in black; T-guards are drawn in blue; O-actions are drawn in purple; the C-guard transitions are drawn in green, which permits to achieve 4 steps of controllability.

Input	Description
wpInBu	*True* when a workpiece is in the input buffer
loc-IL	*True* when the portal is at the drop location of indexed line
loc-In	*True* when the portal is at the input buffer location
loc-CL	*True* when the portal is at the drop location of compact line
pos-U	*True* when the portal is in its up end position
pos-D	*True* when the portal is in its down end position
p-2-IL	*True* when the current work piece should be brought to indexed line
p-2-CL	*True* when the current work piece should be brought to compact line
wpMC	*True* when the workpiece reaches the expected position in front of the machine
MC-done	*True* when the machine finishes its machining
wpCOut	*True* when the workpiece reaches the output position of compact line
wp-Output	*True* when the command of outputting the workpiece is received
wpCLBu	*True* when a workpiece is in the buffer of compact line
pos-E-PC	*True* when the pusher of compact line is in its extended position
pos-R-PC	*True* when the pusher of compact line is in its retracted position

Output	Description
P-G	activate the gripper
P-U	move the portal upwards
P-D	move the portal downwards
P-R	move the portal to the right (towards IL)
P-L	move the portal to the left (towards CL)
BC-P	run the belt of compact line in positive direction
MC	run the machine of compact line
PC-F	move the pusher of compact line forwards
PC-B	move the pusher of compact line backwards

Table 2. Table of inputs & outputs for the portal, belt and machine on the compact line

Initially, the maximum length of the shortest path between any couple of states is 5. Although this seems a reasonable value for controllability, the DTT approach can help to reach a better performance. After adding 5, 12 or 36 C-guards, the maximum length of the shortest path is reduced respectively to 4, 3 or 2 steps.

Now, with the modified specification models, full SIC-testability, full observability, and better controllability is achieved.

4.3 Applying test case reduction with plant features

According to the relations between inputs discussed in Sec. 3.3, for the case study, a set of exemplary plant models are presented in Fig. 7.

The first model in Fig. 7 presents a *mutual exclusion* relation between two input signals: in a nominal behavior, *pos-U* and *pos-D* should not be true at the same time, since a portal cannot be simultaneously in its *up* and *down* position. Similarly, a model for the horizontal movement can be found.

The second model presents a relation among input and output signals. The input *pos-U* remains *True* unless the output *P-D* is activated (and *P-U* is not active). At the same time it is stated that - reading the model from right to left - *pos-U* will not instantaneously but eventually be *True* when *P-U* is activated. Note that it is explicitly

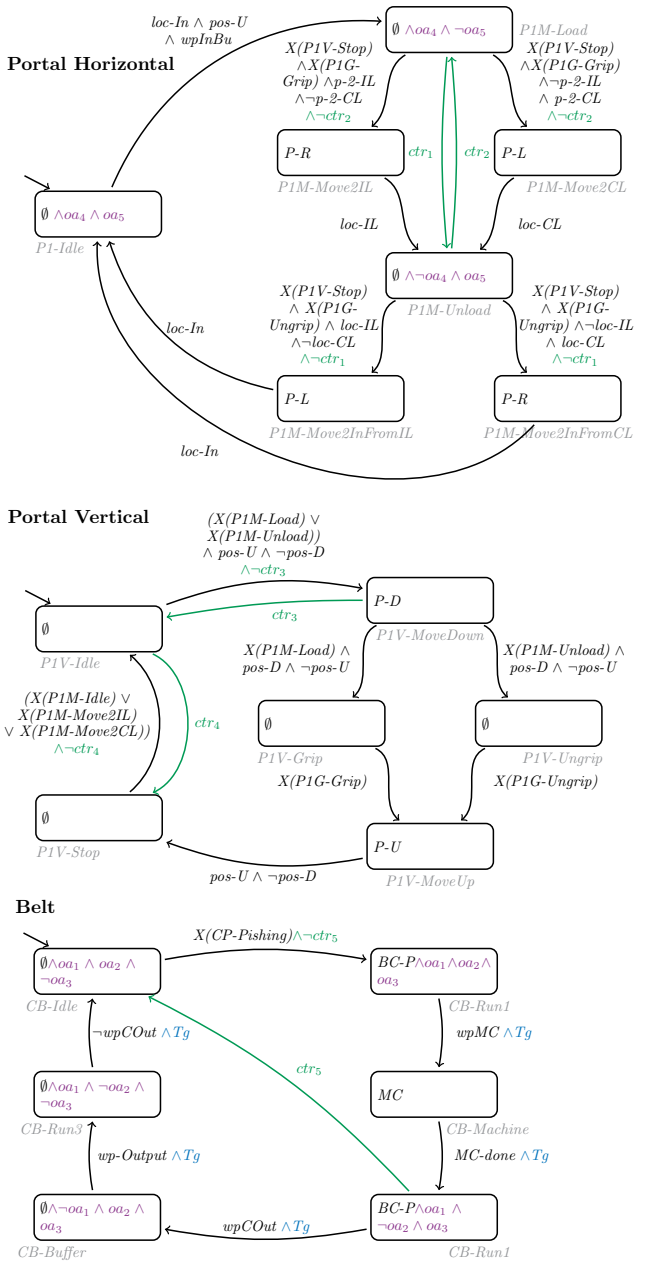

Fig. 6. Specification models for the horizontal portal movement, the vertical portal movement and the belt of the compact line

not stated that *pos-U* will be true directly after the portal movement has been activated, i.e. from location *P1-NotUp2*, *pos-U* will first remain *False* (as in location *P1-NotUp1*), and will eventually become *True* (as in location *P1-Up*). Analogously, a feature for the down movement can be found.

The third model presents a *premise* relation between two input signals. The input *MC-done* can only become *True* when the input *wpMC* is *True*, which means the machine does only operate when the workpiece is at the expected position in front of the machine.

It is worth mentioning that modeling of plant features is based on domain knowledge from engineers. Then, the rest process from composition of specification models to the

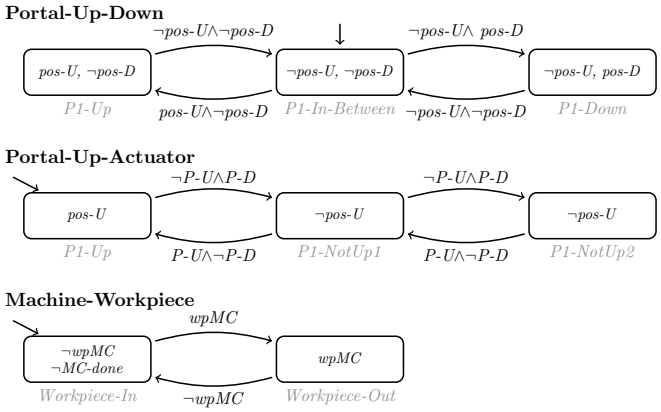

Fig. 7. Plant models for the nominal behavior of the vertical and horizontal portal movement and the machine on the compact line

generation of test sequences are all executed by the tool automatically. In addition, even some simple fragments of the nominal behavior of the system under test contribute to the reduction of test cases; of course, the more plant features are modeled, the higher reduction is obtained.

When the test generation is performed on the SCA from the original specification models solely with CCT approach, a test sequence of 9,647,120 steps is obtained. In the case study, ten plant models are used. As a final result, a test sequence with 448,752 steps is obtained, which leads to a reduction rate of 95% in comparison to the one obtained without plant features.

It can be stated that integrating knowledge about signal relations into the generation process drastically reduces the length of generated test sequences.

5. CONCLUSION

In this work, we presented the current version of our test case generator implementation. The objective of conformance testing is to determine the capability of a software product to adhere to standards, conventions and regulations. In this context, testing of nominal behavior by considering plant features can be a good supplement for complete testing or a replacement when complete testing is not feasible.

In future works, we would like to omit the monolithic composition, as this limits the complexity of the case studies that can be involved drastically. Although we achieved to improve our implementation, the computation is still primary memory intensive. Even though the resulting size is reduced due to the plant features, those extra models have to be considered during calculation, which leads to the question whether and how modular approaches can be applied and to what extent they reduce the computational effort.

Currently, only binary signals are taken into account for the control logic. This crucially restricts the applicability of the presented tool. Consequently, further investigation on extending the capability of handling integer signals is needed. Input equivalence class partitioning might be fruitfully applied. Furthermore, temporal and timing relations between signals can be formulated similar to mutual exclusion and premise.

REFERENCES

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

Enoiu, E.P., Sundmark, D., and Pettersson, P. (2013). Model-based test suite generation for function block diagrams using the UPPAAL model checker. In *IEEE 6th Int. Conf. on Software Testing, Verification and Validation Workshops*, 158–167.

Guignard, A. and Faure, J.M. (2014). A Conformance Relation for Model-Based Testing of PLC. In L. Jean-Jacques (ed.), *12th Int. Workshop on Discrete Event Systems*, 412–419. Cachan.

Jordan, C., Ma, C., and Provost, J. (2017). An educational toolbox on supervisory control theory using matlab simulink stateflow – from theory to practice in one week. In *8th IEEE Global Engineering Education Conference (EDUCON 2017)*.

Kormann, B. and Vogel-Heuser, B. (2011). Automated test case generation approach for PLC control software exception handling using fault injection. In *37th Annual Conf. of the IEEE Ind. Elect. Soc.*, 365–372.

Lee, D., Member, S., and Yannakakis, M. (1996a). Principl Finite S. 84(8).

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

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.

Ma, C. and Provost, J. (2017). A model-based testing framework with reduced set of test cases for programmable controllers. In *13th IEEE Conference on Automation Science and Engineering (CASE)*, 944–949.

Mani, P. and Prasanna, M. (2016). Automatic Test Case Generation for Programmable Logic Controller using Function Block Diagram. In *Int. Conf. on Information Communication and Embedded Systems (ICICES)*, 1–4.

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. doi:10.1016/j.conengprac.2010.10.001.

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 Transactions on Industrial Informatics*, 10(3), 1696–1704. doi:10.1109/TII.2014.2315972.