Extension of the Plant Feature Approach Introducing Temporal Relations

Claudius V. Jordan+, Judit Cuezva Herrero* and Julien Provost+

Abstract— **Testing of controllers with a large number of inputs and outputs suffers from the curse of dimensionality in terms of the combinatorial explosion of possible input combinations. In particular for conformance testing, this results in unbearable large numbers of test cases. To mitigate this problem, the plant feature approach has been proposed, which guarantees full coverage of the *nominal* behavior of the controller. This *nominal* behavior corresponds to the reachable state space when combining in a closed-loop the controller with its fault-free plant. Plant features are identified as intuitive basic knowledge about a system based on its physical behavior. Considering such physical limitations allows to reduce the scope of testing to the actual relevant behavior.**

I. Introduction

With the ever growing demand for more complex systems, offering more functionalities, developed in shorter time, and requiring frequent reconfiguration and update, the control units for these systems also grow larger and gain in complexity. One obvious factor for complexity is the number of input and output signals to be handled by these control units. A direct consequence of the large number of inputs to a system is the curse of dimensionality, which is also known as combinatorial explosion. Nevertheless, engineers desire high confidence about their controllers' behavior. This is why several methods have been proposed and applied in industry to address this problem, such as pair-wise testing, random testing and various model-based approaches [1][2][3].

On the one hand, model-based approaches have in common that they are based on specification models of the system-under-test and derive test cases and test sequences from those specification models. Various methods can be found in the literature based on state machines [4][5] and UML diagrams [6] among others. As stated by [7] still a challenge in MBT based on FSMs is the generation of state verification or identification sequences. An example for a model-based approach to generate such sequences is complete conformance testing, a black-box testing methodology, which takes specification models as basis and evaluates the consequences of every combination of input signals for all states. When comparing this method with others, it presents the advantage that every possible behavior defined in the specification is tested. However, this method suffers from the combinatorial explosion. On the other hand, pair-wise and random testing can be performed without any knowledge about the system because only a subset of all possible input combinations is chosen. For pair-wise testing, all combinations of input pairs are considered, whereas random testing uses any (desired) number of arbitrarily chosen input combinations. The focus for those approaches is the selection of the most relevant test cases given that, due to the combinatorial explosion of the system-under-test's behavior and the finite test execution time, it is not feasible to test all possible input combinations. However, they do not provide a guarantee that all relevant behavior is tested.

Recently, an approach termed "testing with plant features" was proposed in [8] [9]. This approach aims at guaranteeing a high test coverage while, at the same time, reducing the testing effort. This is motivated by the observation that a controller and a physical system (plant) operating in a closed-loop influence each others' behavior, which leads to the conclusion that the reachable state space of the controller is also restricted by the plant it controls. In particular, this is interesting for reactive systems where such features are rather obvious and easy to capture from the physical plant and its structural conditions. An example could be a production automation site. Opposed to approaches like presented in [3], the specifications, which are given as parallel Moore machines, are composed to one monolithic stable composed automaton.

This paper extends the testing with plant features approach proposed in [8]. Its contribution are the definition of *temporal* plant features that allow considering more details about a plant's behavior. These new plant features permit to further restrict the meaningful reachable state space of the controller operated in a closed-loop with the plant. The *temporal* plant features allow specifying relations about sequences of signals, which are added to the bundle of possibilities to focus on the *nominal* behavior of the closed-loop system during testing.

The remainder of this paper is organized as follows. First, the extensions to the existing plant features are presented in Sec. II. Afterwards, two case studies are presented to discuss the application of those extensions in Sec. III. Finally, a discussion on the potential gain during test case generation concludes this paper.

II. Extensions to existing plant features

The plant feature approach has been introduced as an idea to include basic and obvious information about the closed-loop behavior of the controller with the plant, in order to reduce the set of possible input combinations for test case generation. Additional relations can be identified, which can

*Judit Cuezva Herrero is a student at Technical University of Munich, Garching bei München, Germany

+Assistant Professorship for Safe Embedded Systems, Technical University of Munich, Garching bei München, Germany {jordan, provost}@ses.tum.mw.de

be integrated to this concept to represent more closely the *nominal* behavior of the system and, consequently, further improving the efficiency of the test case generation and execution.

This section focuses on such aspects that have not been taken into account in the plant feature approach so far. The characteristic presented in this contribution are the *temporal* plant features. *Temporal* relations represent information regarding the order in time.

A. Temporal Signal Relations

The first extension to the existing plant features is the temporal relationship which occurs between signals. Such a temporal relation is, for example, a *sequence* of observed sensor values as consequence of activated actuators, which is currently not taken into account within the plant feature approach.

The former plant features were capable of defining static relations between two signals (input-input or output-input). Given N inputs, in general 2^N possible combinations have to be considered in every state. For example, given two signals a and b, 2^2 combinations are possible at each time step, which are $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$.

When formulating static relations such as *premise* and *mutual exclusion*, one out of four combinations can be omitted [9]. These static relations can be seen in Fig. 1.

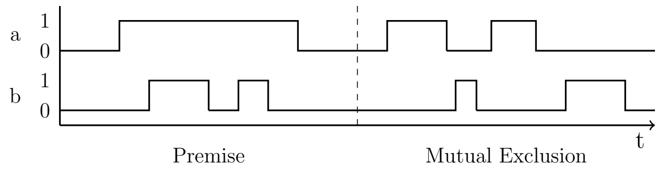

Fig. 1. Evolution of two signals a and b over time. There are two situations displayed. (a) Signal a is a premise for signal b, (b) Signals a and b are mutually exclusive.

In addition to the static relations, the newly introduced features take into account sequences of input combinations. Consequently, the number of possible sequences of input combinations for k time steps for N boolean inputs is $(2^N)^k$. By introducing a new *temporal* plant feature, the feasible input sequences can be reduced.

As an example, in a system one signal has to become 1 and fall back to 0 before another one will eventually become 1 (and reset to 0). Some sequences of signal combinations will not represent the nominal behavior of the system. This is interesting for the test case generation in terms of reduction of combinatorial complexity.

In Fig. 2, possible combinations of signals over time are displayed: *premise*, *strict sequence* and *overlapped sequence*. The main difference between those are the order of signal changes highlighted by blue ellipses. The premise (which is also a static relation) represents the situation that a signal b can only be 1 when another signal a is also 1 at the same time. Otherwise, b cannot be 1. For a strict sequence, a signal a has to be 1 and reset to 0 before another signal b can do so. Another change of signal a is possible but it has no effect unless b has been set to 1 and reset to 0. Compared to the strict sequence, in the overlapped sequence b has to occur as long as signal a is 1 and at the same time a is not allowed to return to 0 until then. Likewise, b must change back to 0 only after a.

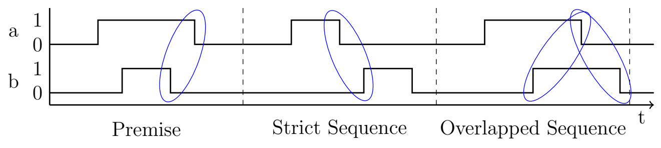

Fig. 2. Evolution of two signals a and b over time. There are three situations displayed. (a) Signal a is a premise for signal b, (b) Signals a and b occur in strict sequence, (c) Signals a and b occur in a overlapped sequence.

B. Temporal Plant Features

The graphical implementation as state machine for the temporal plant features of two signals a and b is depicted in Fig. 3. The values of signals a and b can be 0 (e.g. $\neg a$) or 1 (e.g. a). This results in a concise description of the input restrictions on the states and transitions that cause changes on the restrictions. On the left side, the strict sequence feature is displayed, and on the right side the feature for overlapped sequence. For both cases, it can be seen that initially b is 0, because b is forced to be 0 represented by $\neg b$ in the state. If a has the value 1, the transition to the next state is fired and the restrictions in the next state (here still $\neg b$) are activated, which has no effect in this case. If afterwards a resets to 0 ($\neg a$), the restrictions in the state are released. $\emptyset$ in a state indicates that there is no restriction on the set of inputs.

For the strict sequence, b stays 0 ($\neg b$ in the states) until a has become 1 and has been reset to 0 because then the restrictions are released ($\emptyset$ in the state). For the overlapped sequence, a (b) has to stay 1 (0) until b (a) has changed its value to 1 (1) and then b is forced to stay 1 at least until a has reset to 0.

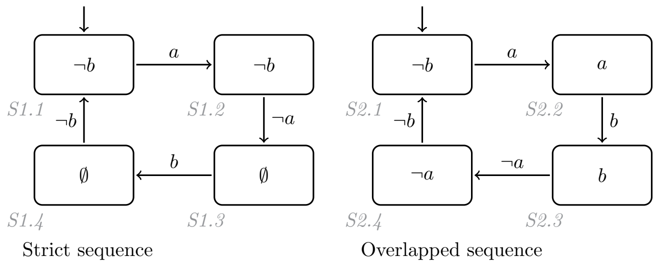

Fig. 3. State machine representation of the temporal plant feature *strict sequence* and *overlapped sequence* for two signals a and b.

The presented temporal extension allows a further decrease in possible input combinations. As stated in the example, it is possible to reduce from originally 4 combinations down to 3 combinations, by means of static plant features. In addition, according to the restrictions imposed on the sequence of combinations, eventually, the input combinations that have to be considered can be reduced to a minimum of 2. This can be seen on states *S1.1* and *S1.2* as $\neg b$ obliges the signal value for signal b to be 0, consequently, in that situation only the two combinations $(a, \neg b)$ and $(\neg a, \neg b)$ are feasible. In comparison, for the static features, e.g. mutual exclusion of signals a and b, only one combination is eliminated.

III. Case Studies

The applicability of the plant feature approach is demonstrated with two case studies, first, a single one-way conveyor belt with two proximity sensors and a pusher (Sec. III-A), and then, a more complex elevator control (Sec. III-B). In particular, the *temporal* plant features introduced in Sec. II are further discussed and applied.

A. Conveyor Belt

The conveyor system consists of a single, one-way belt which has two proximity sensors, one situated at the beginning of the belt and the other one at the end. The pusher situated at the end of the conveyor belt exports the package from the system. The system is depicted in Fig. 4 and its inputs and outputs are listed in Tab. I.

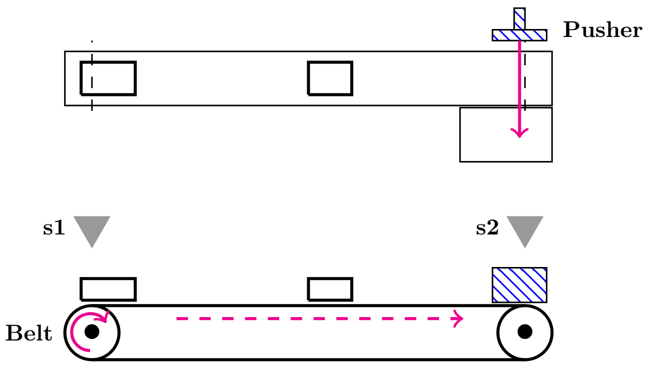

Fig. 4. Figure representing the conveyor system used in the case study. The top image represents the conveyor belt seen from above, whilst the bottom image shows the lateral view of the system.

TABLE I
Table of Inputs & Outputs for the Conveyor Belt

Input	Description
s1, s2	Sensor detecting packages at certain positions on the belt. *s1* is positioned at the beginning. *s2* is located at the end.

Output	Description
Belt	Move the belt.
Pusher	Push package out of the system.

One can easily identify *premise* relations in the system for the belt movement (output signal *Belt*) and the sensors *s1* and *s2*: changing values of sensor *s1* from 1 to 0 and values of sensor *s2* from 0 to 1.Likewise, the movement of the pusher (output signal *Pusher*) is a premise for resetting sensor *s2* to 0.The reason for such plant feature is that the sensors' values will only change when the belt is running (resp. the pusher is moving).If the belt is not running, then no package can leave the position of sensor *s1*.The output signal *Belt* is a premise for the change of sensor value *s1* from 1 to 0.The same reasoning can be applied for a package reaching the position of sensor *s2*. The output signal *Belt* is again a premise for the change of the sensor value *s2* from 0 to 1.Similarily, the change of sensor value *s2* from 1 to 0 depends on the pusher.Note that the change from 0 to 1 for sensor *s1* is not dependent on the movement because the upstream process (e.g. a crane) can deploy a package at this position at any time.

However, temporal relationships between signals have not yet been considered. It can be seen that a package is going to be detected first by sensor *s1* and later by sensor *s2*, which means that a temporal relationship exists between them. As discussed in Sec. II, a plant feature can be formulated to represent such situations. An implementation as state machine is given in Fig. 5, where the sequence of possible states considering only one package in the system at a time are drawn in black color. When there is only one package allowed in the system, the strict sequence plant feature is straightforwardly instantiated from the template provided in Fig. 3. Labels for output signals that enable the transition from one state to another are added.

The transitions are again labeled with guards and actions. If there is no guard, then the transition is independent of an output signal, which is indicated by "-". The states are labeled with a tuple representing the current signal values of the sensors (*s1*, *s2*). The only possible sequence is a strict sequence of the input combinations $(\neg s1, \neg s2)$, $(s1, \neg s2)$, $(\neg s1, \neg s2)$ and $(\neg s1, s2)$. Compared to the reduction induced by the three premises identified beforehand, this temporal plant feature further reduces the set of input combinations varying over time.

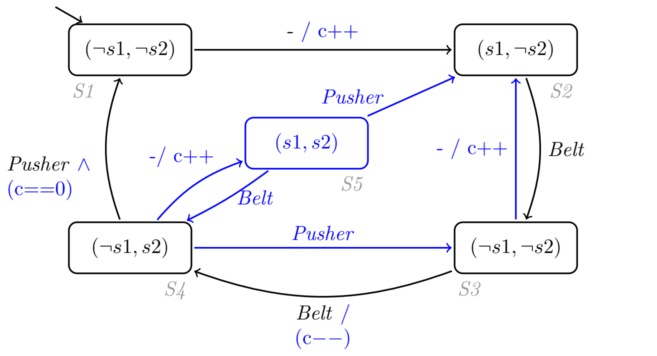

Fig. 5. Temporal plant feature for the conveyor belt example for the signal values of sensors *s1* and *s2*. Without the blue addition (edges and state $(s1, s2)$) only one package is considered in the system. The blue addition (introducing a counter *c*) allows considering multiple packages in the system at the same time.

When allowing more than one package at a time, the proposed feature template is less intuitive. An advantage of the plant feature approach is, that once the features are specified, they can be gathered in a library such that a user picks a template and the implementation is done automatically. The additions in blue color in Fig. 5 make it feasible to consider multiple packages in the system at the same time.

Therefore, a counter *c* is added as internal variable. Now, the state $S5$ is also reachable, as there can be a previous package detected by sensor *s2* and a new one entering the system at sensor *s1* simultaneously. This is in particular of interest as it is considered not *nominal* when sensor *s2* would encounter more packages than sensor *s1*. Actions, here increment (c++) and decrement (c−−) the counter *c*, are specified on the transitions. That way, the states represent the current combination of sensor values, which is of interest for the test case generation, and the guards represent restrictions on the change of those input combinations. For example, when currently the sensor input combination is $(s_1, \neg s_2)$ (state *S2*), the change to the combination $(\neg s_1, \neg s_2)$ is considered nominal according to the plant feature if the belt is running, indicated by the output signal *Belt*.

For the extended feature displayed in Fig. 5, a modified version can be formulated, omitting the particular initial state with no package in the system by merging it with the situation that there is currently no package at neither of the two sensors. This modification is presented in Fig. 6.

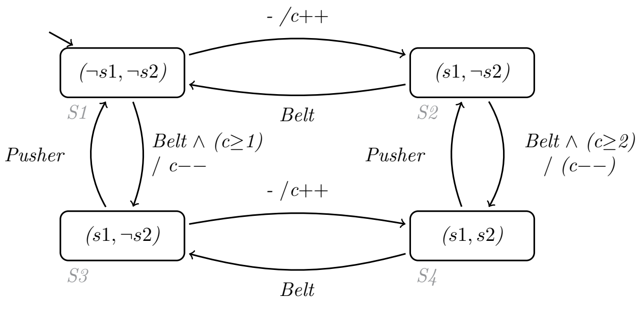

Fig. 6. Modified temporal plant feature for the conveyor belt based on Fig. 5

B. Elevator

The elevator in this case study serves three different floors: level 1, 2 and 3. On each floor there is a sensor indicating the presence of the elevator cabin *(s1, s2, s3)*, as well as door sensors to indicate when the doors on the floors are closed *(dc1, dc2, dc3)* or when they are open *(do1, do2, do3)*. Additionally, the elevator has internal doors which also have door sensors *(dci, doi)*. For safety reasons, the doors of each floor must always be closed, unless the elevator is on that floor. The internal door is opened only when resting at one of the floors. The vertical movement of the elevator is represented by the output signal *MoveU* for the upwards movement and *MoveD* for moving downwards. The opening and closing of doors is also done by separate motors, controlled by the output signals *OpD1, OpD2, OpD3, OpDi* for opening and *ClD1, ClD2, ClD3, ClDi* for closing. Additionally, there is a presence sensor *(pres)*, which detects obstacles in the doors' way, and an overweight sensor *(overw)* indicating excessive weight in the cabin. The elevator system is depicted in Fig. 7 and the system's inputs and outputs are listed in Tab. II.

Note, the elevator moves from floor to floor based on user requests expressed via the buttons in the cabin or the ones on each floor. The elevator's movement cannot be interrupted by any other call until it has finished the previous movement, which means that if the elevator is going from the third to the first floor, it won't stop on the second floor on its way down.

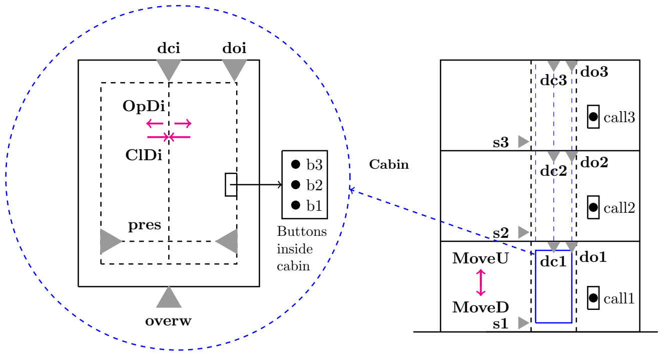

Fig. 7. Elevator with inputs and outputs.

TABLE II
TABLE OF INPUTS & OUTPUTS FOR THE ELEVATOR

Input	Description
b1, b2, b3	Buttons inside the cabin, to demand which floor to go to.
s1, s2, s3	Sensors indicating the cabin's position on a specific floor.
dci, dc1, dc2, dc3	Sensors indicating that the doors are closed when value 1 (internal doors, doors on floor 1, 2, 3).
doi, do1, do2, do3	Sensors indicating that the doors are open when value 1 (internal doors, doors on floor 1, 2, 3).
overw	Overweight sensor indicates overweight.
pres	Presence sensor indicates that something is detected in the doors' way.
call1, call2, call3	Button on each floor to enable calling the elevator.

Output	Description
MoveU	Move the elevator upwards.
MoveD	Move the elevator downwards.
OpDi, OpD1, OpD2, OpD3	Open doors (internal doors, doors on floor 1, 2, 3).
ClDi, ClD1, ClD2, ClD3	Close doors (internal doors, doors on floor 1, 2, 3).

Several static plant features can be identified in the elevator system. General mutual exclusion is represented in Fig. 8. A special type of mutual exclusion, the so-called "chain"-exclusion, can be found for the elevator that can physically only be at maximum one floor at a time, which can be captured in a mutual exclusion feature of the sensor values *s1, s2* and *s3*. In addition to the mutual exclusion, the order of possible signal combinations is fixed, which allows to make use of the findings regarding temporal relations presented in Sec. II. In Fig. 9 the state machine model for this feature is presented. Note that this differs from the one presented in Fig. 8.

A simple example for a premise feature can be the relation between the internal doors (*dci*) not being closed and

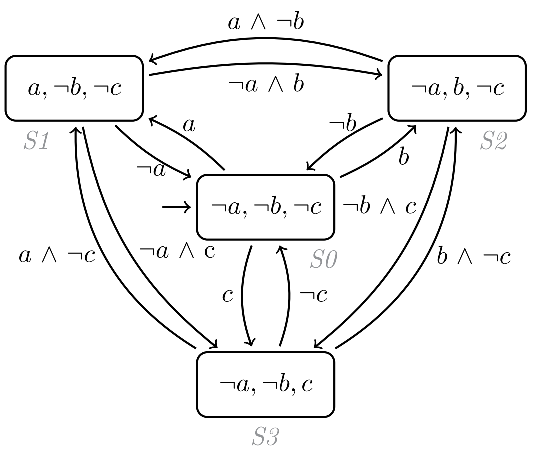

Fig. 8. General mutual exclusion. When direct transition between the leafs is not possible, the feature is called "star"-exclusion.

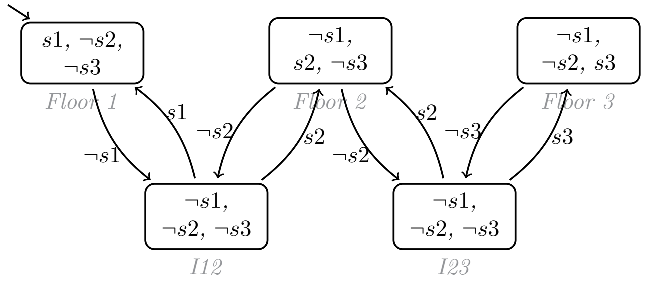

Fig. 9. Mutual exclusion of the floor sensors of type "chain"-exclusion.

detecting overweight (*overw*). If the doors are open, there can be an overweight detected or not, however, once the doors have been closed, the overweight sensor will not change its value. This premise relation is depicted in Fig. 10.

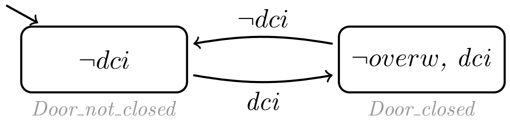

Fig. 10. Open internal door (*dci*) is a premise for detecting overweight (*overw*).

Additionally, new plant features presented in Sec. II can be found. For the mutual exclusion example mentioned above regarding the floor sensors, a temporal relationship can be added depending on the elevator's movement and the sequence of inputs which could be received. It looks quite similar to the feature presented in Fig. 9 as the states represent again the position of the cabin. However, the feature presented in Fig. 12 is a level-2 feature. Note, as introduced in [9] level-1 features refer to relations between input signals only, whereas level-2 deals with output signals. For example, if the elevator is in the third floor (*s3*) and it is moving downwards *MoveD* to the first floor, then *s3* will deactivate, *s2* will activate and deactivate, until finally *s1* will activate. The signal changes over time are represented in Fig. 11. The same sequence, in inverse order, applies if the elevator is moving upwards. The state machine for this plant feature (upwards and downwards movement) is presented in Fig. 12.

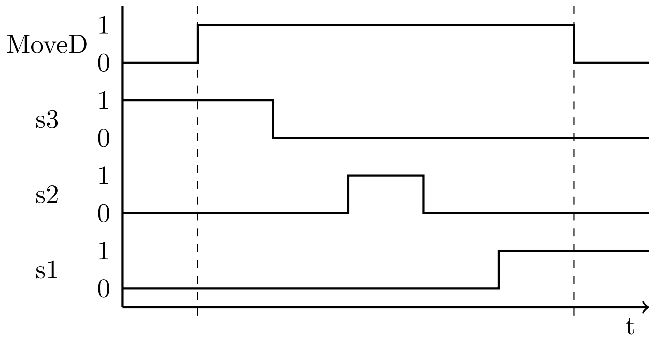

Fig. 11. Evolution of floor sensor signals indicating the position of the elevator cabin when moving downwards from the third to the first floor.

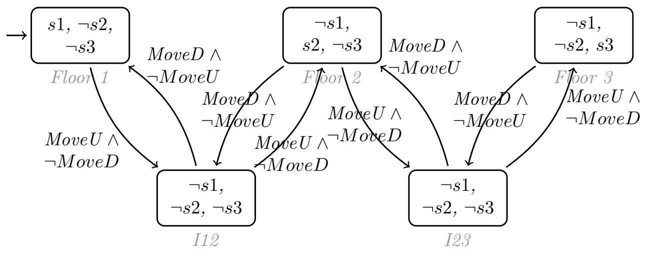

Fig. 12. Temporal relationship between the floor sensors (*s1*, *s2*, *s3*) as a chain depending on the elevator's movement (*MoveU*, *MoveD*).

By applying this sequence to the plant features approach, it avoids the generation of inputs which would not match the real behavior of the system, i.e. activating the third floor sensor and then directly the first floor sensor, without the activation of the second floor in between.

TABLE III
MEALY MACHINE RESULTS OBTAINED FOR THE ELEVATOR CASE STUDY

Case Considered	**# states**	**# transitions**	**Reduction in %**
CCT	575	301465600	–
with static plant features	323	9506322	96.8%
with temporal plant features	323	7594660	97.5%

Automatic test case generation is based on Mealy machines that are generated based on a specification model. In order to compare the effort to perform complete conformance testing (CCT), the sizes of the generated Mealy machines for the different cases discussed above are given in Tab. III. When the restrictions expressed via plant features are considered, the number of states and transitions is greatly reduced. For the case study presented here, 12 static plant features could be identified. With those static plant features the number of transitions in the generated Mealy machine could already be reduced by 96.8%. Identifying that one of the mutual exclusion features is a “chain”-exclusion, as discussed above, leads to further reduction by additional 20% resulting in an overall testing effort that is less than 3% compared to CCT.

IV. Conclusion & Discussion

In this paper, the plant feature approach has been enriched by considering temporal relations. As temporal relations, a strict sequence and an overlapped sequence have been introduced. In addition, the mutual exclusion has been generalized to more than two signals and applied to the presented approach. The ”Chain”-Type mutual exclusion is one useful plant feature resulting from combining the before-mentioned relations. As a result, further test cases can be neglected during testing as they would not occur in the actual nominal closed-loop behavior of the controller and the physical plant under control.

Two examples have been presented that show the applicability of the newly introduced plant features, where for the second quantitative results are provided. Further case studies will have to be conducted to show how much reduction in terms of test cases can be achieved applying these additional features. For future work it will be interesting to incorporate timing in the specification model as well as using timing information in a new set of plant features.

References

[1] W.-l. Huang and J. Peleska, “Model-based testing strategies and their (In)dependence on syntactic model representations,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9933 LNCS, pp. 3–21, 2016.

[2] P. Mahadik, D. Bhattacharyya, and H. jin Kim, “Techniques for automated test cases generation: A review,” *International Journal of Software Engineering and its Applications*, vol. 10, no. 12, pp. 13–20, 2016.

[3] K. Pinkal and O. Niggemann, “A new approach to model-based test case generation for industrial automation systems,” in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*. IEEE, jul 2017, pp. 53–58.

[4] A. C. Pinheiro, A. Simão, and A. M. Ambrosio, “FSM-based test case generation methods applied to test the communication software on board the ITASAT university satellite: A case study,” *Journal of Aerospace Technology and Management*, vol. 6, no. 4, pp. 447–461, 2014.

[5] R. Dorofeeva, K. El-Fakih, S. Maag, A. R. Cavalli, and N. Yevtushenko, “FSM-based conformance testing methods: A survey annotated with experimental evaluation,” *Information and Software Technology*, vol. 52, no. 12, pp. 1286–1297, dec 2010.

[6] Y. D. Salman and N. L. Hashim, “Automatic Test Case Generation from UML State Chart Diagram: A Survey,” 2016, pp. 123–134.

[7] A. Sabbaghi and M. R. Keyvanpour, “State-based models in model-based testing: A systematic review,” in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*. IEEE, dec 2017, pp. 0942–0948.

[8] C. Ma and J. Provost, “Using plant model features to generate reduced test cases for programmable controllers,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 11 163–11 168, jul 2017.

[9] ——, “A model-based testing framework with reduced set of test cases for programmable controllers,” in *13th IEEE Conference on Automation Science and Engineering (CASE)*, 2017, pp. 944–949.