Buzhinsky, Igor; Vyatkin, Valeriy

Modular plant model synthesis from behavior traces and temporal properties

Published in:
Proceedings of the 22nd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2017

DOI:
10.1109/ETFA.2017.8247578

Published: 04/01/2018

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Buzhinsky, I., & Vyatkin, V. (2018). Modular plant model synthesis from behavior traces and temporal properties. In *Proceedings of the 22nd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2017* (Vol. Part F134116). (Proceedings IEEE International Conference on Emerging Technologies and Factory Automation). IEEE. https://doi.org/10.1109/ETFA.2017.8247578

Modular Plant Model Synthesis from Behavior Traces and Temporal Properties

Igor Buzhinsky[1, 2], Valeriy Vyatkin[1, 3]
[1] Department of Electrical Engineering and Automation, Aalto University, Finland
[2] Computer Technology Department, ITMO University, St. Petersburg, Russia
[3] Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Sweden
{igor.buzhinskii, valeriy.vyatkin}@aalto.fi

***Abstract*—Reliability of industrial automation software, which is usually ensured with testing and simulation, can be improved using formal analysis and, in particular, the technique of model checking. In model checking, considering the closed-loop composition of the plant model and the controller model allows checking a larger class of properties than in the more traditional open-loop case, where the model of the controller is verified alone. Constructing the formal model of the plant automatically may significantly reduce human workload and mitigate the human factor issue. Commonly, complex industrial plants and controllers have modular structure, and thus the problem of automatic construction of a modular plant model is important. This paper proposes two techniques which extend an earlier proposed method of monolithic plant model construction to the modular case.**

I. Introduction

Testing and simulation are commonly used to establish safety and correctness of industrial automation systems. It is widely known, however, that these techniques are not exhaustive, which in practice leads to the trial-and-error approach, which may cost money and human lives. In contrast, the technique of model checking [1], [2] can prove or refute relevant system requirements by performing a complete state space analysis of the system's formal model.

Model checking of control logic is commonly applied in open loop, where this logic is considered independently from the plant with which it operates. For example, open-loop model checking has been adopted in the Finnish nuclear industry [3], [4]. Another approach, which is adopted in the present paper, is closed-loop model checking [5], where the *plant* is considered explicitly. This facilitates more comprehensive analysis.

In closed-loop approaches, the plant is traditionally modeled manually [5], [6]. Nevertheless, automated approaches to plant model construction have also been proposed [7]–[9]. Such approaches reduce human workload and the related human factor. Except [8], where plant models are built to be used for anomaly detection and not model checking, these approaches are monolithic: that is, the plant model is constructed as a single finite-state machine [10]. According to [11], [12], such approaches are only suitable for small systems. This situation is not common for applications where model checking is important. Moreover, the work [12] reviews approaches to formal verification which are based on modularity and use it to reduce the computational complexity of formal analysis.

This paper proposes two modular plant model construction techniques, further developing the framework introduced in [9]. As input for these techniques, behavior traces and temporal properties are employed. The first technique is essentially a graph algorithm which combines basic plant models built according to the approach [9] into a larger model, at the same time utilizing available behavior examples. The second technique independently combines basic plant modules, forming their synchronous composition. Both techniques are evaluated on a case study from the nuclear power plant (NPP) industry.

The rest of the paper is organized as follows. Sections II and III present related work and concepts used throughout the paper. Then, Section IV introduces techniques to extend the previously proposed plant model construction method [9] to the case of a modular system. These techniques are evaluated in Section V on a case study. In Section VI, concluding remarks and thoughts are given.

II. Related work

Several works consider automatic plant model generation. The work [9], which is used as the basis for this paper, presents a method of monolithic plant model construction based on Boolean satisfiability (SAT) solvers. Its contribution will be discussed in more detail in Section III-C. Two other works [7], [13] do not support modularity, and hence we do consider them in detail. On the contrary, the works [5], [6] describe modular techniques, but they suggest constructing plant models manually, using a set of generic basic modules.

In [8], probabilistic deterministic timed automata are generated based on simulation traces. This approach supports modularity and is capable of automatically determining the modular structure of the system based on the communication network of the automation system. The model of modularity in [8] assumes that different components of the systems are asynchronous and operate in parallel. Moreover, they can be linked by output and input variable connections. Unlike [8], the present paper considers synchronous operation of system components. Connections of components by variables is also not supported. However, in the present paper plant models are generated in the way suitable for formal verification and not anomaly detection, like in [8].

III. PRELIMINARIES

This section starts by defining several concepts and introducing several tools used in the paper. Then, it proceeds with definitions which will be widely applied in Section IV.

A. Model checking

Model checking [1], [2] is a formal verification technique which exhaustively explores the state space of the system's model and checks requirements stated in formalisms such as linear temporal logic (LTL) or computation tree logic (CTL). These requirements are called *temporal requirements*, or *temporal properties*, since they may involve *temporal operators*, which relate states of the model in different time instants. For example, using temporal operators $\mathbf{G}$ ("always"), $\mathbf{X}$ ("next") and $\mathbf{F}$ ("in the future"), LTL property $\mathbf{G}(\mathbf{X}\, f \rightarrow \mathbf{F}\, g)$ conveys the textual requirement "always, if f is satisfied on the next step, then g shall be satisfied at least once in the future."

In the context of automation systems, *open-loop* and *closed-loop model checking* [5] are distinguished. While in open-loop model checking the controller is verified regardless of the plant with which it operates, in closed-loop model checking the plant is also modeled. A comparison of closed-loop and open-loop model checking is performed in [14]. According to [14], these approaches complement each other. Hence, if the plant model is available, both approaches can be applied, and verification capabilities are broader.

B. Used tools

In this paper, three software tools will be employed. Apros[1] is a continuous process simulator, which is suitable for modeling both the plant and the controller automation logic in the form of block diagrams. It will be applied to record behavior traces, which will be used as input data for plant model construction. Then, NuSMV[2] [15] is a symbolic model checker. While explicit-state model checkers, such as SPIN and UPPAAL, analyze the state space of the model using graph algorithms, symbolic model checkers represent the state space implicitly using Boolean encodings of initial states and allowed state transitions. This allows processing significantly larger state spaces. NuSMV models are represented in a textual format. Finally, the tool MODCHK [16] brings Apros and NuSMV together: it allows creation of formal models as block diagrams, like in Apros, and is able to convert them to the NuSMV format for further verification.

C. Plant model synthesis

According to [11], approaches to plant model construction are subdivided into monolithic and modular ones, and only the modular approach, where the overall plant model is composed of smaller models, is applicable in real cases.

The techniques of modular plant model construction described in the present paper are based on the work [9], where a method to synthesize monolithic plant models as nondeterministic finite-state machines has been presented. This method is based on SAT solvers, which places some restrictions on the size of input the method can handle and on the size of generated models. These restrictions are caused by the computational complexity of the SAT problem and the resultant need to apply heuristics to solve it. Nevertheless, in [9] (Section III-C) a largely simplified technique has also been proposed. The difference between these two techniques (from now on, the *SAT-based* and the *simplified* techniques, respectively) is in the support of *LTL properties* as input data, whereas *behavior traces* can be used in both techniques.

[1] http://www.apros.fi/en/

[2] http://nusmv.fbk.eu/

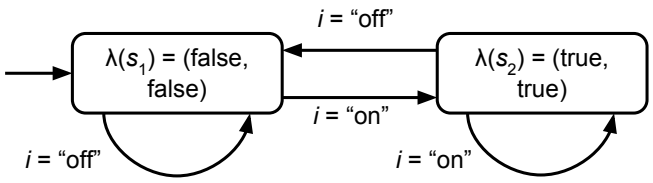

Fig. 1. Example of a plant model representing a system of two lights. In this model, $S = \{s_1, s_2\}$, $S_0 = \{s_1\}$ (the state on the left), $\mathcal{I}$ contains a single input representing the light control signal, $I = \{\text{"on"}, \text{"off"}\}$, $\mathcal{O}$ contains two Boolean outputs showing whether each of the lights in on, $O = \{\text{true}, \text{false}\}^2$, T is represented with arrows, and λ is shown inside states.

D. Plant and controller models

We will represent plant models using the formalism of nondeterministic finite-state machines, like it was done in [9]. Such entities are easily representable in the language of NuSMV, which we will use in the case study (Section V).

First, a plant model has a set of *inputs* $\mathcal{I} = \{\iota_1, ..., \iota_k\}$, where each input $\iota_j, 1 \leq j \leq k$ belongs to a finite, non-empty set of values $\mathcal{I}_j$. For example, $\mathcal{I}_j$ may represent the set of Boolean values, a set of integers, or a partition of a continuous range into a number of contiguous intervals. The Cartesian product $I = \mathcal{I}_1 \times ... \times \mathcal{I}_k$ will be called the set of *input events*. Similar notations will be used for the set of *outputs* $\mathcal{O} = \{\omega_1, ..., \omega_m\}$, corresponding values sets $\mathcal{O}_j, 1 \leq j \leq m$, and the set of *output events* $O = \mathcal{O}_1 \times ... \times \mathcal{O}_m$.

Then, a *plant model* is a tuple $P = (S, S_0, \mathcal{I}, I, \mathcal{O}, O, T, \lambda)$, where S is the finite set of *states*, $S_0 \subset S$ is the non-empty set of *initial states*, $\mathcal{I}$, I, $\mathcal{O}$ and O are the sets of inputs, input events, outputs and output events defined above, $T \subset S \times I \times S$ is the *transition relation*, and $\lambda : S \rightarrow O$ is the *output function*. A plant model must adhere to the completeness, or the absence of deadlocks, requirement: $\forall s_1 \in S, i \in I : \exists s_2 : T(s_1, i, s_2)$, which means that the model is able to respond to any possible input event. This definition complies with the one from [9] except the definition of input and output events (called inputs and outputs in [9]). An example of a plant model represented graphically as a state machine is provided in Fig. 1.

Finally, a *controller model* is defined similarly to the plant model, except that the output function has a different domain: $\lambda : S \times I \rightarrow O$. In other words, while plant models are Moore machines [10], controller models are Mealy machines [10]. Commonly, controller models are deterministic, but this assumption is not required for the purpose of this paper.

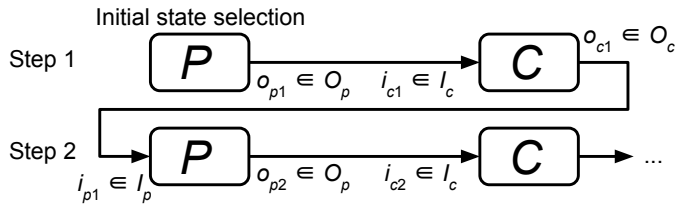

Fig. 2. The first two steps of plant and controller model interaction.

E. Model compositions

Given two plant models P_1 and P_2 with input and output sets $\mathcal{I}_1, \mathcal{O}_1$ and $\mathcal{I}_2, \mathcal{O}_2$ respectively, the composition P of these plant models has the set of inputs $\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2$ and the set of outputs $\mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2$. Currently, we will not place any more constraints on plant model compositions, but instead we will consider two concrete ways to construct such compositions in Section IV.

The restriction on inputs and outputs of the composition C of controller models C_1 and C_2 is identical to the one above. On the other hand, since controller models are the ones to be verified, their composition must produce outputs obtained as the result of the execution of controller components. In this paper, due to the synchronous operation of controller subsystems in Apros, we will assume that C is a synchronous product of state machines representing C_1 and C_2. Other definitions may be also considered, but the exact definition of a controller composition does not need to be fixed in the context of this paper.

Finally, assume that a number of plant models $P_1, ..., P_p$ and a number of controller models $C_1, ..., C_c$ are composed into single models P and C respectively, whose input and output sets are named I_p, I_c, O_p and O_c. Assuming also that $I_c \subset O_p$ and $I_p \subset O_c$, the closed-loop composition L is a state machine having no inputs and the output set $O = O_p \cup O_c$. A single cycle of this state machine is performed as follows: the output event of C is converted to the input event of P (this can be done by truncating redundant input values for P), P makes a transition, the output event of P is converted to the input event of C, and, finally, C makes a transition. Thus, P and C execute in turns. On the first turn, instead of receiving the event from the controller, P selects its initial state without making a transition. The described procedure is illustrated in Fig. 2.

F. Input specification for plant model construction

Traces, or behavior examples, and LTL properties are the sources of input data for plant model construction in [9]. Assume that I and O are input and output event sets of the plant model to be synthesized (which in our case will consist of several components). A *trace element* is a pair (o, i), where $o \in O$ and $i \in I$. Then, a *trace* of length l is a sequence $((o_1, i_1), ..., (o_l, o_l))$. Such sequences may be obtained by running a simulation model of the considered automation system (e.g. in Apros) and recording the values of relevant system parameters.

LTL properties is the second type of input data which the resulting plant model must satisfy. LTL properties are composed of Boolean connectives (e.g. $\wedge$, $\vee$, $\neg$), temporal operators ($\mathbf{X}$, $\mathbf{F}$, $\mathbf{G}$, $\mathbf{U}$) and atomic propositions:

- $\texttt{input}(i), i \in I$: input event i was received by the plant on the current cycle of the plant-controller interaction;
- $\texttt{output}(o), o \in O$: output event o was produced by the plant on the current cycle of the plant-controller interaction.

IV. Modular plant model synthesis

Using the approach proposed in the work [9], it is possible to synthesize plant models complying with the definition from Section III-D (with minor denotational difference) directly as nondeterministic finite-state machines. We will call such models *monolithic* ones. In this section, two techniques of *modular* plant model construction are proposed. They generate resulting plant models in the NuSMV format.

On the first stage, both techniques assume that the modular structure of the plant and the controller is available. In particular, suppose that the plant model P to be constructed should be formed of modules $P_1, ..., P_p$, whose input and output sets are not required to be disjoint. Initially, both techniques construct the models of each of $P_1, ..., P_p$ using the methods presented in [9] (either the SAT-based or the simplified ones). It remains to compose these models, and this is the aspect where the proposed plant model construction techniques differ. The differences are explained below.

A. Explicit-state composition

Assume that we wish to construct the composition $P = (S, S_0, \mathcal{I}, I, \mathcal{O}, O, T, \lambda)$ of two plant models $P_1 = (S_1, S_{01}, \mathcal{I}_1, I_1, \mathcal{O}_1, O_1, T_1, \lambda_1)$ and $P_2 = (S_2, S_{02}, \mathcal{I}_2, I_2, \mathcal{O}_2, O_2, T_2, \lambda_2)$. If more than two models need to be composed, this can be done by first composing the first two models, then composing the result with the third model, and so on.

First, we need to introduce several auxiliary definitions. Input events $i_1 \in I_1$ and $i_2 \in I_2$ are *consistent*, if all their components which correspond to matching inputs from $\mathcal{I}_1$ and $\mathcal{I}_2$ are identical. Input event $i = i_1 \cdot i_2 \in I$ for P is a *composition* of two consistent input events i_1 and i_2, if its components are formed of the components of i_1 and i_2, and duplicate components are taken only once. Similar definitions are assumed for output events. For states, the notation $s = s_1 \cdot s_2$ means that s is such that $\lambda(s) = \lambda_1(s_1) \cdot \lambda_2(s_2)$.

The explicit-state composition is based on the idea of constructing the Cartesian product of individual state machines P_1 and P_2 with additional filtering based on traces. Hence, the corresponding algorithm is a variant of the breadth-first search (BFS) algorithm. Its pseudocode is presented in Alg. 1 and is explained below.

A queue q of composite states, and a set v of processed composite output events are maintained (they are declared on lines 1–2). Initially (lines 3–9), the queue is filled with the pair of initial states of P_1 and P_2 which are consistent with

Algorithm 1: Explicit-state plant model composition. Construction of $\mathcal{I}$, I, $\mathcal{O}$, O and λ is not shown.

Data: plant model $P_1 = (S_1, S_{01}, \mathcal{I}_1, I_1, \mathcal{O}_1, O_1, T_1, \lambda_1)$, plant model $P_2 = (S_2, S_{02}, \mathcal{I}_2, I_2, \mathcal{O}_2, O_2, T_2, \lambda_2)$, trace set TR
Result: plant model $P = (S, S_0, \mathcal{I}, I, \mathcal{O}, O, T, \lambda)$

```
1  q ← empty queue;
2  v ← ∅;
3  for s1 ∈ S01, s2 ∈ S02 do
4      if consistent(s1, s2) then
5          add s1 · s2 to S0;
6          enqueue s1 · s2 to q;
7          add λ1(s1) · λ2(s2) to v;
8      end
9  end
10 while q is not empty do
11     dequeue s = s1 · s2 from q;
12     for t1 = (s1, i1, s1') ∈ T1, t2 = (s2, i2, s2') ∈ T2 do
13         o1' ← λ1(s1');
14         o2' ← λ2(s2');
15         if consistent(i1, i2) ∧ consistent(o1', o2') then
16             if o1' · o2' is present in TR then
17                 s' ← s1' · s2';
18                 add t = (s, i1 · i2, s') to T;
19                 if o1' · o2' ∉ v then
20                     add o1' · o2' to v;
21                     add s' to S;
22                     enqueue s' to q;
23                 end
24             end
25         end
26     end
27     for i1 ∈ I1, i2 ∈ I2 do
28         if consistent(i1, i2) ∧ ∄x : T(s, i1 · i2, x) then
29             add t = (s, i1 · i2, s) to T;
30         end
31     end
32 end
```

each other. Then, until the queue becomes empty, the loop on lines 10–32 is executed. The next composite state s, which originates from individual states s_1 and s_2, is taken from the queue (line 11). For each pair of transitions outgoing from s_1 and s_2 in P_1 and P_2, if (1) the input events i_1 and i_2 of these transitions are consistent with each other, (2) in P_1 and P_2 they lead to states s_1' and s_2' with consistent output events o_1' and o_2', and (3) the composite output event $o' = o_1' \cdot o_2'$ occurs in traces at least once, then a new composite state s' (associated with o') is created with the corresponding transition from s to s' labeled with $i_1 \cdot i_2$ (lines 17–18). If output event $o_1' \cdot o_2'$ has not been previously encountered, s' is registered as a new state and added to the queue (lines 19–23). After state s has been processed, it may happen that there are no outgoing transitions from s for some composite input events. In such a situation, the completeness requirements would be violated. Thus, missing transitions are added as self-loops (lines 27–31).

In addition, we need to mention the issue of trace support of transitions. A transition of a plant model labeled with input event i_1 from a state labeled with output event o_1 to a state labeled with output event o_2 is *supported*, if the pair of contiguous trace elements $((o_1, i_1), (o_2, i_2))$, where i_2 is an arbitrary input event, occurs in the set of traces at least once. The knowledge of whether transitions are supported or not may be used in temporal specifications to be checked for the closed-loop system. Unsupported transitions are less reliable and are only added to satisfy the completeness requirement. Hence, ignoring them may prevent "false alarms" during model checking, while still allowing some faults to be revealed. In basic plant models, unsupported transitions are added as self-loops in the simplified plant model construction method, and in a more elaborate way in the SAT-based method. In the composite plant model, a transition t composed of two individual transitions t_1 and t_2 is labeled as unsupported if and only if either of t_1 or t_2 is unsupported. The transitions added on line 29 of Alg. 1 are also annotated as unsupported.

A simple example of running the algorithm can be imagined in the context of Fig. 1. Assume that we wish to compose two plant models, each representing a control system for one of the two lights. If P_1 and P_2 both accept the same input, behave according to Fig. 1, but each of them has only one output, then the result P of their explicit-state composition is exactly the model presented in Fig. 1.

Finally, several properties of the described algorithm should be mentioned. As visible from line 16, the algorithm does not add states which correspond to composite output events absent in traces, and this bounds the size of S with the number of distinct output events in traces. As for transitions, they can be added even if they are absent in traces.

Then, the particular feature of the SAT-based method [9] is the support of LTL properties formulated for the plant model. Assuming that P_1 and P_2 are constructed in compliance with some LTL properties, the composite model will also comply with them, possibly except unsupported transitions added on line 29, which can be excluded from consideration during model checking. Thus, the major benefit of the described algorithm is the possibility to construct plant models using both behavior traces and LTL properties for the situations of larger plant model and trace set sizes than it was possible in [9].

B. Purely modular composition

For the purely modular composition, we consider all the plant models $P_1, ..., P_p$ to be composed simultaneously into a single model P. Whereas in the explicit-state approach a specific synchronous composition of state machines was considered, in the purely modular composition models $P_1, ..., P_p$ are executed synchronously in parallel, independently from each other. If some inputs of $P_1, ..., P_p$ overlap, then this means that some inputs for P are passed to more than one of the state machines $P_1, ..., P_p$. In case of an output overlap,

the situation is more difficult, since the outputs of different basic plant models may be conflicting. In this case we allow any possible combinations of individual output events, thus introducing an additional source of nondeterminism into the composite plant model. However, a shortcoming of this approach is the inability to maintain the compliance of P with the LTL properties which hold for $P_1, ..., P_p$. This problem does not arise when all outputs of $P_1, ..., P_p$ are disjoint. Similarly to the explicit-state composition algorithm, the purely modular algorithm can be applied to the example of two light control systems, and the resulting model is again exactly the one from Fig. 1.

V. Evaluation on a case study

In this section, the proposed techniques of modular plant model synthesis are evaluated on a case study from the NPP industry. The experiments in this section were performed on the Intel Core i7-4510U CPU with the clock rate of 2GHz.

A. Case study system

The proposed techniques were evaluated on the case study based on the Apros model of an NPP with a pressurized water reactor (PWR), from now on called the generic PWR model. This model was provided by Fortum Power and Heat Oy,[3] a power utility with NPP operation license in Finland.

We focused on two subsystems of this model, namely the reactor and turbine trip subsystem and the pressurizer pressure control subsystem. These subsystems were modeled in MODCHK and then converted to the NuSMV language. From the plant side, several plant components related to these subsystems were selected: the upper plenum, the reactor, live steam and the pressurizer. The overview of the considered modular structure is provided in Fig. 3. In this figure, not all possible component inputs and outputs are shown. For controller components, several inputs were not related to the selected plant components and hence were assumed to take any values in allowed intervals (like in open-loop model checking). For plant components (which will be further composed using the techniques proposed in this paper), several signals from the controller were ignored as entirely or partially duplicating other signals and at the same time making the overall model more complex. Moreover, plant components may be indirectly influenced by controller components which are beyond the case study. Such dependencies were not considered as well. The purpose of the described simplifications was to allow formal verification to terminate in realistic time and to understand the results better.

Another point to notice is that in Fig. 3 all inputs and outputs of plant components are disjoint. This means that in the explicit-state composition algorithm (Alg. 1) the function "consistent" always returns "true", and that the purely modular algorithm (Section IV-B) is able to compose basic plant models while preserving LTL property compliance, like the explicit-state one.

[3]http://www.fortum.com/

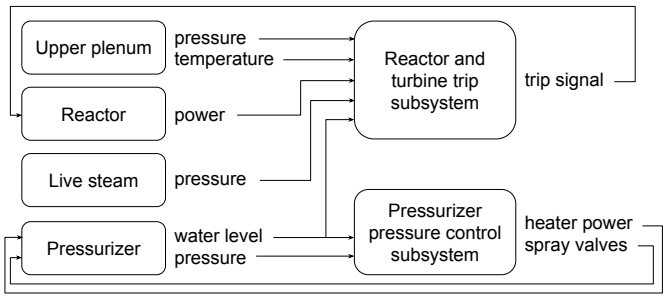

Fig. 3. Modular structure of the case study. Plant modules are shown on the left, and controller modules are shown on the right.

B. Trace recording

All the parameters mentioned in Fig. 3 were recorded during the trace collection procedure in Apros. In total, 3000 traces were recorded, each trace having the length of four minutes. Traces were discretized in time with the step of one second, meaning that each trace comprised 240 elements. Each continuous parameter was discretized into several intervals (*discrete levels*). To make the recorded traces different from each other, the simulation for each of the traces was started with different initial plant conditions (e.g. different water levels in tanks). In addition, small alternations (e.g. setpoint value changes) were randomly applied to the controller to make the plant exhibit unwanted behaviors during simulation. The presence of such behaviors in traces facilitates the construction of a more complete plant model.

Unfortunately, the high computational complexity of the SAT-based method did not allow us to use all the collected traces as input data. The most crucial issue was RAM consumption, and hence the number of traces was reduced to keep it within 4 GB. As a result, only 50 randomly selected traces were used as input data for the SAT-based method.

C. Plant model construction and composition

Basic plant models (i.e. the ones on the left in Fig. 3) were prepared using both the SAT-based and the simplified methods. "Smoothness" LTL properties were used for the SAT-based method: for each output, a condition was added specifying that the discrete level of this output could not change faster than this was possible according to traces. For example, assume that output x has three possible discrete levels, and Boolean formulas x_1, x_2, x_3 express that the value of x belongs to each level. Then the smoothness condition restricting immediate level changes of x by one can be expressed as follows: $\mathbf{G}((x_1 \rightarrow \mathbf{X}(x_1 \vee x_2)) \wedge (x_2 \rightarrow \mathbf{X}(x_1 \vee x_2 \vee x_3)) \wedge (x_3 \rightarrow \mathbf{X}(x_2 \vee x_3)))$.

In the simplified method, no LTL properties were used since this method does not support them. However, the LTL properties described above were chosen in a way that plant models constructed by this method are satisfied automatically: all transitions in the model are either taken from traces or added as self-loops. As a result, models constructed this way can be regarded as more reliable ones: they are built based on

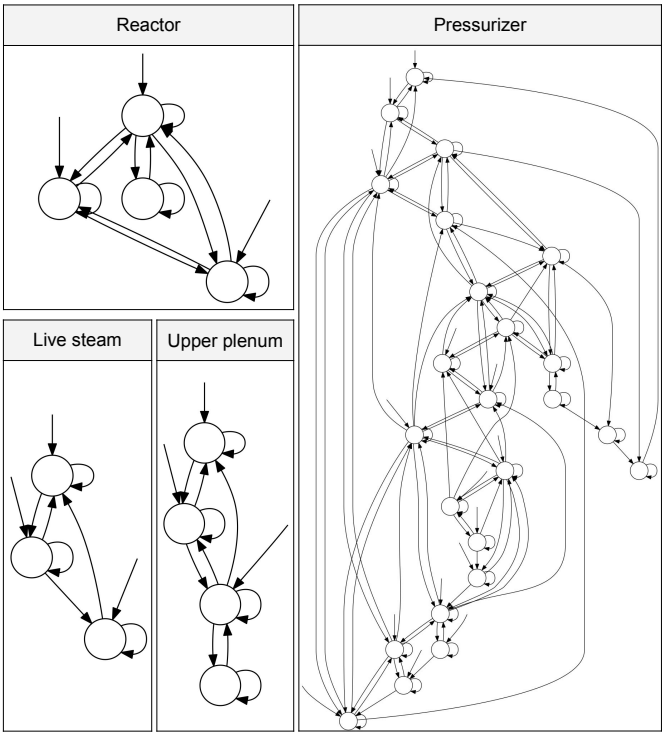

Fig. 4. Basic plant models composed into a purely modular model.

a larger set of traces. Despite this, both methods constructed the models of the same size: 4, 3, 3 and 24 states for the upper plenum, the reactor, live steam and the pressurizer models respectively. The whole set of models was constructed in 384 and 42 seconds in cases of SAT-based and simplified model construction respectively.

Then, two plant model construction methods from Section IV were executed for each of two combinations of basic plant models, resulting into four composite models in total. In particular, explicit-state compositions were generated in 5 seconds with 97 states (for basic models constructed by the SAT-based method) and in 32 seconds with 207 states (for basic models constructed by the simplified method). Obtaining purely modular composition is instant since the procedure is limited to a proper arrangement of basic NuSMV models specified textually. Fig. 4 shows an overview of the generated purely modular model in the case of basic models generated by the simplified method.

D. Closed-loop model checking

Each of four composite plant models was composed with the model of the controller (consisting of two modules) in a closed loop. All the models were represented in the language of the NuSMV verifier. Then, a set of 43 CTL properties was model checked for each model. These properties expressed the original requirements for controller modules and also the attempts of the authors to understand the closed-loop behavior of the original simulation model. For example, for the pressurizer pressure control subsystem, these requirements consisted of request-response properties to adjust actuators according to sensor measurements, properties forbidding spurious actuation of such adjustments, and properties requiring the pressure in the pressurizer to restore if it exceeds or falls below certain values. Properties of the latter type were formulated in such a way that unsupported transitions were excluded from consideration. In addition, fairness constraints were added in NuSMV to prevent infinite execution of self-loops, which represent unrealistic plant behaviors.

Model checking times are reported in Table I. As visible from the table, explicit-state models require more time in model checking than purely modular ones. This may be connected with the complexity of their NuSMV representations, which influences the performance of symbolic model checkers such as NuSMV. Next, plant models composed of basic plant models built according to the SAT-based approach were the most difficult ones to verify. A possible reason for this is the more complex positioning of transitions in these models than in the ones built according to the simplified approach, where unsupported transitions are added as self-loops.

As for model checking results, they were identical for almost all of the plant models, except the SAT-based explicit-state model, for which one of the requirements was satisfied which was violated for other plant models. This requirement, which is only meaningful in the closed-loop context, expressed that the reactor relative power eventually becomes low if the live steam pressure falls below a certain value.

E. Comparison with original monolithic methods

To show the necessity of modular techniques, original SAT-based and simplified methods were run to construct the overall system in a monolithic way, with all inputs and outputs from each plant components simultaneously. The simplified method was able to construct this model in 72 seconds, which is approximately the amount of time required to run the explicit-state modular method assuming that basic plant models also need to be constructed. The number of states in this model was equal to 207, similarly to the modular case: this is the number of distinct output events in traces. However, we again note that the simplified method is unable to handle LTL properties.

Then, we attempted to run the SAT-based method in the same setting. To keep RAM consumption below 4 GB, we had to further reduce the number of traces to 10. With this number of traces, the method constructed a model with 46 states in 33 minutes. Thus, it is visible that modular plant model construction techniques suggested in this paper are advantageous with respect to the original SAT-based monolithic method in the case when the support of LTL properties is important.

VI. Discussion and conclusions

In this paper, two modular plant synthesis techniques have been proposed. For basic plant model synthesis, the method from [9] has been adopted. Then, while the explicit-state technique (Section IV-A) constructs a single finite-state machine from basic plant models, the purely modular technique

TABLE I
MODEL CHECKING TIMES

Composite plant model	Time (minutes)
SAT-based, explicit-state	1730.5
SAT-based, purely modular	371.4
Simplified, explicit-state	31.4
Simplified, purely modular	3.3

(Section IV-B) simply combines unchanged basic models into a synchronous composition.

The techniques have been evaluated on a case study, and model checking results show high coherence between different types of modular models. The purely modular technique has been found superior in terms of the time required for model checking by the employed symbolic verifier NuSMV. However, the techniques can also be applied together with an explicit-state verifier (such as UPPAAL or SPIN). In this case the explicit-state technique, which produces plant models with smaller state spaces, would instead reduce verification time.

Next, only the purely modular technique actually preserves the modular structure of the composite model. Such a structure makes the generated models easier to comprehend. On the other hand, assuming that basic plant models were constructed to be compliant with LTL properties, combining basic plant models with overlapping outputs may cause violation of these LTL properties for the composite model. A similar problem exists in the explicit-state approach: some transitions which are absent in traces must be added to the composite model to maintain the completeness requirement. These transitions may also lead to LTL property violations, and disabling them is currently possible only in temporal specifications checked for the closed-loop system.

Finally, both techniques produce plant models with different extents of trace generalization, with purely modular models showing more diversity in behavior. Also accounting for specific benefits and shortcomings of each of the technique, we believe that it is reasonable to apply both of them in closed-loop verification.

We must note that the performed case study has several limitations. First, in the modular structure of the involved system all input and outputs of basic plant models were disjoint, which did not allow us to consider how the explicit-state technique handles parameter compliance between different basic models. Then, the abilities of the authors to understand the obtained results were restricted by the limited knowledge of the used generic PWR model. Finally, closed-loop model checking has only been performed using a symbolic verifier.

An additional result of this paper, which is not directly connected with the proposed techniques, is related to evaluating the SAT-based basic plant model construction method proposed in [9]. It has been shown that the computational limitations of this method regarding RAM prevent it from being applied on large trace sets. Hence, the much more simple method of plant model construction from traces only may be more applicable in practice if LTL property support is not crucial.

Future work may involve considering basic plant models with connected inputs and outputs and asynchronous models, like in [8]. The mentioned limitations of the case study should also be mitigated.

ACKNOWLEDGMENTS

This work was financially supported by the SAUNA project (funded by the Finnish Nuclear Waste Management Fund VYR as a part of research program SAFIR2018), by the Engineering Rulez project funded by Tekes – the Finnish Funding Agency for Innovation (Decision 3095/31/2016), and by the Ministry of Education and Science of the Russian Federation, project RFMEFI58716X0032.

The authors thank Antti Pakonen from VTT Technical Research Centre of Finland for preparing the formal model of the reactor and turbine control subsystem in MODCHK.

REFERENCES

[1] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.

[2] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[3] J. Lahtinen, J. Valkonen, K. Björkman, J. Frits, I. Niemelä, and K. Heljanko, "Model checking of safety-critical software in the nuclear engineering domain," *Reliability Engineering & System Safety*, vol. 105, pp. 104–113, 2012.

[4] A. Pakonen, J. Valkonen, S. Matinaho, and M. Hartikainen, "Model checking for licensing support in the Finnish nuclear industry," in *International Symposium on Future I&C for Nuclear Power Plants (ISOFIC)*. Korean Nuclear Society, 2014.

[5] S. Preuße, *Technologies for Engineering Manufacturing Systems Control in Closed Loop*. Logos Verlag Berlin GmbH, 2013, vol. 10.

[6] J. Machado, B. Denis, and J.-J. Lesage, "A generic approach to build plant models for DES verification purposes," in *8th International Workshop on Discrete Event Systems (WODES)*. IEEE, 2006, pp. 407–412.

[7] M. Roth, L. Litz, and J.-J. Lesage, "Identification of discrete event systems: Implementation issues and model completeness," in *7th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, vol. 3, 2010, pp. 73–80.

[8] A. Maier, A. Vodencarevic, O. Niggemann, R. Just, and M. Jaeger, "Anomaly detection in production plants using timed automata," in *8th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2011, pp. 363–369.

[9] I. Buzhinsky and V. Vyatkin, "Automatic inference of finite-state plant models from traces and temporal properties," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1521–1530, 2017.

[10] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: A cyber-physical systems approach*. Lee & Seshia, 2011.

[11] M. Perin and J.-M. Faure, "Building meaningful timed models of closed-loop DES for verification purposes," *Control Engineering Practice*, vol. 21, no. 11, pp. 1620–1639, 2013.

[12] S. Tripakis, "Compositionality in the science of system design," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 960–972, 2016.

[13] H.-T. Park, J.-G. Kwak, G.-N. Wang, and S. C. Park, "Plant model generation for PLC simulation," *International Journal of Production Research*, vol. 48, no. 5, pp. 1517–1529, 2010.

[14] J. Machado, B. Denis, and J.-J. Lesage, "Formal verification of industrial controllers: with or without a plant model?" in *7th Portuguese Conference on Automatic Control (CONTROLO)*, 2006, pp. 341–346.

[15] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: a new symbolic model checker," *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 4, pp. 410–425, 2000.

[16] A. Pakonen, T. Mätäsniemi, J. Lahtinen, and T. Karhela, "A toolset for model checking of PLC software," in *18th IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, 2013, pp. 1–6.