Scalable Methods of Discrete Plant Model Generation for Closed-Loop Model Checking

Igor Buzhinsky, Antti Pakonen, Valeriy Vyatkin

Citation:


DOI: https://doi.org/10.1109/IECON.2017.8216949

Publisher’s statement:

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Scalable Methods of Discrete Plant Model Generation for Closed-Loop Model Checking

Igor Buzhinsky1, 2, Antti Pakonen3, Valeriy Vyatkin1, 4

1 Department of Electrical Engineering and Automation, Aalto University, Finland
2 Computer Technology Department, ITMO University, St. Petersburg, Russia
3 VTT Technical Research Centre of Finland Ltd, Finland
4 Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Sweden

igor.buzhinskii@aalto.fi, antti.pakonen@vtt.fi, vyatkin@ieee.org

Abstract—To facilitate correctness and safety of mission-critical automation systems, formal methods should be applied in addition to simulation and testing. One of such formal methods is model checking, which is capable of verifying complex requirements for the system’s model. If both the controller and the controlled plant are formally modeled, then the variant of this technique called closed-loop model checking can be applied. Recently, a technique of automatic plant model generation has been proposed which is applicable in this scenario. This paper continues the work in this direction by presenting two plant model construction approaches which are much more scalable with respect to the previous one, and puts this work into a more practical context. The approaches are evaluated on a case study from the nuclear automation domain.

I. INTRODUCTION

Reliability is a crucial requirement for industrial automation systems. As a traditional approach of verification and validation (V&V), testing is widely applied. However, certain mission-critical systems such as aerospace and nuclear instrumentation and control (I&C) ones require stronger guarantees of correctness and safety. Among well-known techniques offering deeper analysis is model checking [1], a formal verification technique which can prove system correctness (or, more precisely, the correctness of the system’s formal model) for the entire range of its possible behaviors.

In the industrial automation context, at least two separate formal models are often considered: plant process model (later referred to as the plant model) and the automation system model (from now on, the controller model). Having only the controller model, one can verify systems in open loop [2]: the controller’s correctness is checked without considering the environment where it is operating. If the plant model is also available, the more natural closed-loop modeling and verification [3]–[6] become possible.

Apart from the actual formal verification, the challenge of formal model construction is crucial: it often requires manual effort, making the prerequisites of model checking harder to achieve and introducing the so-called human factor, a major source of errors. For plant models in particular, a model constructed manually might be too detailed and thus difficult to verify, and adjusting it to make verification feasible would require even more effort. Thus, approaches have been proposed to construct plant models automatically [6], [7].

This paper builds on top of the work [6] and presents two plant model construction methods whose main advantage over [6] is the degree of scalability, which was limited in [6] due to the use of SAT solvers. As input data, we use system traces collected with the help of a simulation model of the system. This choice puts plant model construction in a more practical context, which is demonstrated by evaluating the proposed methods on a case study based on a nuclear power plant (NPP) simulation model. Then, tuning model generation parameters such as the size of the trace set and the strength of discretization enables finding the right trade-off between the precision of the model and the computational complexity of its verification.

The paper is structured as follows. Section II introduces concepts used throughout the paper. Then, Section III describes the proposed plant model generation techniques. In Section IV, the techniques are evaluated on a case study. The results are discussed in Section V.

II. PRELIMINARIES

A. Simulation environments

Specialized simulation environments aid the construction of automation systems and allow simulating them. Typically, these environments present means of constructing both the controller model (that is, the entity whose correctness must be ensured), and the plant model, with which the controller operates. In this paper, the Apros1 continuous process simulator is used. In Apros, the automation system can be represented as a number of function block diagrams, either expressing control logic or the plant’s mechatronic aspects.

B. Discrete formal models

While simulation models are created for simulation and testing, formal models are intended to be applicable for formal verification. They are often represented in finite-state formalisms, such as finite-state machines [8], timed automata [9] and Petri nets [10]. These kinds of formal models are applicable for verification by means of model checking, which is described in Section II-C. Simulation models, in contrast, cannot be directly applied in model checking: they typically contain

1http://www.apros.fi/en/
continuous parameters which are not adequately supported by modern model checkers.

Both the controller and the plant can be represented not only by simulation models, but also by formal models. According to [11], formal plant models are subdivided into detailed and abstract ones: while detailed models preserve the internal structure and parameters of the plant, abstract models are more simple and only convey its external behavior. Plant model construction in the present paper is based on traces showing external behavior, so the inferred models are abstract.

C. Model checking

Model checking [1] is a formal verification technique which exhaustively explores the state space of the system’s formal model to check certain requirements, which may involve statements over the model’s state at different moments of time. The most formal languages used to formulate such temporal properties are the linear temporal logic (LTL) [12] and the computation tree logic (CTL) [13]. They operate with the simplest, logical model of time: it is measured as the number of executed state transitions, or discrete steps. The problem of model checking is, given a discrete finite-state model and a temporal requirement, to determine whether this requirement is satisfied for this model, and, if it is not, optionally provide a counterexample trace.

In CTL, constraints over the system’s formal model are expressed with Boolean connectives and temporal operators. For example, operators $\text{AG}$, $\text{AF}$ and $\text{AX}$ express that their argument is satisfied always, eventually or on the next step of the behavior respectively for all future behaviors of the model. Corresponding operators $\text{EG}$, $\text{EF}$ and $\text{EX}$ formulate the same requirements for at least one future behavior.

D. Model checking of automation systems

Model checking of automation systems can be performed in either open loop or closed loop [3]–[6]. The work [14] references several concrete open-loop and closed-loop approaches. While the closed-loop approach explicitly considers the model of the plant, in open-loop model checking, a more traditional and simple approach, the plant model is reduced to plant sensor measurements which are assumed to have arbitrary values and are independent from each other. This can result in counterexample behaviors that demonstrate scenarios that are not possible in the real world. Such scenarios can, for example, involve measurements radically changing their values in an instant, which is impossible for physically slow systems. Ruling out such counterexamples manually costs effort.

On the other hand, nuclear automation systems are designed to be fault tolerant, and an analyst should not automatically disregard a counterexample that shows “impossible” plant response since it may be relevant due to possible failures of plant sensors. Thus, from the safety point of view, closed-loop modeling represents a challenge. By closing the loop, the state space of the model is usually reduced [3], and it is possible that relevant behaviors (that would reveal a design issue) are accidentally filtered out [15].

A comparison of open-loop and closed-loop model checking is provided in [16]. According to this work, these approaches are complementary to each other. Then, in closed-loop model checking it is possible to check temporal properties which involve plant variables unobservable by the controller. Model checking results of other properties may be different in the open-loop and closed-loop cases.

E. Model checking tools

Two model checking tools are involved in the present study. The first tool is the symbolic model checker NuSMV [17]. It accepts the model and the specification to be checked in the form of a text file. The model has a number of variables with values from finite ranges. The initial state of the model and allowed state changes are expressed with constraints which form two formulas, $\text{INIT}$ and $\text{TRANS}$. While $\text{INIT}$ simply defines which value combinations of state variables form valid initial states, $\text{TRANS}$ is formulated over two variable set: the set of current variables, denoted by variables names, and the set of next-step variables, whose names are additionally marked by the keyword next.

By means of such constraints, expressing finite-state machines becomes straightforward in NuSMV. The second involved tool, called MODCHK, is the enhancement of the toolset presented in [18] and can be viewed as a user-friendly interface to NuSMV which allows to specify and verify automation system as a number of function block networks. MODCHK is also partially integrated with Apros.

F. Related work

In [7], simplified simulation plant models are synthesized based on source code of PLC programs. Formal modeling is not considered in [7], and the construction method works only for manufacturing systems. In [19], formal discrete-state models of the entire closed-loop system are constructed as nondeterministic finite-state event generators using the data obtained from a running real-world controlled plant. This approach, however, is not suitable for mission-critical systems such as NPPs, which must be verified prior to operation. The technique [20] constructs automation system models to detect anomalies, but not for formal verification. In [5] and [21], plant models are built based on 3D CAD drawings, but the construction process is not fully automated. Another field of research is construction of software models [22].

Finally, the work [6] deals with automatic formal plant model construction from behavior examples and LTL specification of the plant. Using such a plant model, it is potentially possible to verify multiple controllers prior to their deployment. However, the method proposed in [6] has limited scalability and cannot construct plant models with hundreds or thousands of states given hundreds or thousands of behavior examples. The present paper develops the work [6], mitigating the scalability issue by focusing on a more specific problem where only behavior examples, or traces, are available.
III. PLANT MODEL CONSTRUCTION

Following some preliminary definitions, two plant model construction methods are proposed in this section. Both approaches are currently implemented to produce NuSMV models, but can be modified to support other formats. Their implementation is available online as a part of the extended finite-state machine construction toolset.2

A. Definitions

The source of data for automatic plant model construction is a set of traces, or input-output behavior examples of the plant. Since the plant model is the part to be constructed automatically, traces will be viewed in terms of its interface. First, the plant model has a set of inputs \( I = \{i_1, \ldots, i_k\} \), each of which can be either Boolean or real-valued (continuous). Inputs describe the state of plant actuators, which receive information from the controller. Second, the plant model has a set of Boolean or real-valued outputs \( O = \{o_1, \ldots, o_m\} \). Together, plant model inputs and outputs will be referred to as plant model parameters. Plant and controller model interaction is cycle-based: on each cycle the controller model first reads plant model outputs and then produces inputs to the plant model. The duration of each cycle is fixed.

A trace element is a pair \((O, I)\), where \( O \) is a list of \( m \) outputs and \( I \) is a list of \( k \) inputs. A raw trace of length \( \ell \) is a finite sequence of trace elements \((\{(O_1, I_1), \ldots, (O_n, I_n)\})\). Since model checking usually operates with discrete values, continuous parameters must be discretized. This is done by distributing the values of each continuous parameter into a finite number of intervals, the indices of which will be referred to as discrete levels. A discretized trace is a sequence \((\{(O_1, I_1), \ldots, (O_n, I_n)\})\) obtained from the raw trace by replacing each real value with the corresponding discrete level. Boolean values remain unchanged (i.e. they have two discrete levels). A discretized trace \((O', I')\) is obtained from the raw trace \((O, I)\) by selecting values from the corresponding discrete set. Discrete levels of plant model outputs. To construct the state machine, where each state corresponds to certain discrete levels of plant model outputs. To construct the state machine, the following procedure, partially adopted from [6], is applied. For each element of \( v(O) \) found in discretized traces, a state is created. This state is included into \( S_0 \) if and only if it corresponds to the beginning of some trace. For each pair of contiguous discretized trace elements \((\{(O_1, I_1), (O_2, I_2)\})\), a transition labeled with \( I_2 \) is added from \( O_1' \) to \( O_2' \).

To prevent deadlocks, if some transitions are missing after the described procedure is applied, they must be added. Differently from [6], the unknown input combination is directed to the same states as the closest (averaged over all inputs) combination for which a transition sourcing in the considered state exists. Depending on whether a transition has been added based on this completion procedure or traces, it will be called either an unsupported or a supported one, respectively. Transitions triggered by input combinations absent in traces (unknown input combinations) are added in a different way: if ordinary transitions are present from \( O_i' \) to \( O_j', \ldots, O_m' \), then a transition group is added for each unknown input combination to each of \( O_j', \ldots, O_m' \). Fig. 2 shows an example of a state machine constructed using the discretized trace from Fig. 1.

Finally, the finite-state machine is explicitly encoded in NuSMV, which is a straightforward process. Discrete levels of real-valued parameters are mapped to corresponding numeric intervals (only integer values are allowed in NuSMV). During the encoding, additional fairness constraints are added to prevent self-loops from being executed externally. Self-loops are ubiquitous in plant models due to the common situation when none of the plant model outputs change their discrete levels between consequent trace elements. Paths eternally taking these loops often prevent target plant states from being reached (since the state of the plant remains unchanged forever), which causes violations of liveness requirements. Fairness constraints remove such behaviors from consideration. In NuSMV, these constraints are introduced by defining a variable indicating that the state of the plant has changed during the last step, and adding a fairness declaration for this variable, obliging it to be true infinitely often.

C. Constraint-based method

The rationale behind the constraint-based, or the symbolic method, was to produce condensed models with shorter NuSMV representations which are easier for NuSMV to process. Instead of explicitly defining states and transitions, this method constrains plant model parameters and their changes between two adjacent steps. In the NuSMV model, a separate

---

2https://github.com/ulyantsev/EFM-tools/
IV. EXPERIMENTAL EVALUATION

In this section, the proposed plant model generation methods are evaluated on a case study. All experiments were performed on the Intel Core i7-4510U CPU with the clock rate of 2 GHz.

TABLE I

<table>
<thead>
<tr>
<th>Type</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INIT out1' in {1, 2}</td>
</tr>
<tr>
<td>2</td>
<td>INIT (\text{out1'} = 1 &amp; \text{out2'} )</td>
</tr>
<tr>
<td>3</td>
<td>TRANS (\text{out1'} = 1 &amp; \text{next(out1') in {1, 2}})</td>
</tr>
<tr>
<td>4</td>
<td>TRANS (\text{out1'} = 1 &amp; \text{next(out1')} = 1 &amp; \text{next(out2')})</td>
</tr>
</tbody>
</table>

Fig. 3. Plant model constructed by the constraint-based method given the trace from Fig. 1 and represented as a state machine. The figure corresponds to the constraints from Table I. Transitions for unknown input combinations are not shown and are assumed to lead to both states from the left state and only to the right state from the right state.

A. Case study

For the case study, a generic simulation model of an NPP with a pressurized water reactor (PWR), hereafter referred to as the generic PWR model, was used. This Apros model was provided by Fortum Power and Heat Oy,\(^3\) a power utility with NPP operation license in Finland, and includes the most important process, mechatronic and control components of an NPP, and corresponding automation logic. Due to its large size, the case study considered only eight automation block networks out of 44 networks modeled in Apros. The names of these eight subsystems were masked and are indicated as S1, ..., S8. Subsystems S1, ..., S5 are responsible for activating protection functions, and the rest control pressures and liquid levels in tanks. Controller NuSMV models for the selected subsystems were constructed using MODCHK.

Functional CTL requirements were prepared for each sub-

http://www.fortum.com/
system using the documentation of the generic PWR model, its actual implementation and our own understanding of the model. **Controller-only requirements** are the ones which are sensible to be verified without a plant model (they do not depend on feedback between the plant and the controller). The most common examples of such requirements are of the request-response type: they require some controller outputs (e.g. an activation of a protection function) given some conditions over inputs. In contrast, **plant-and-controller requirements** are not intended for open-loop verification as they restrict plant model outputs which are assumed to be unrestricted in open-loop verification. For example, the following plant-and-controller requirements were formulated for S7:

1) $\text{AG}(p < p_1 \rightarrow \text{AF} p > p_1)$: “always, if pressure $p$ goes below $p_1$, eventually it will be above $p_1$” for several low pressures $p_1$;

2) $\text{AG}(p > p_2 \rightarrow \text{AF} p < p_2)$: “always, if pressure $p$ goes above $p_2$, eventually it will be below $p_2$” for several high pressures $p_2$.

Simulating the generic PWR model, we collected 3000 traces, each with 240 elements corresponding to a four minute long simulation with the sampling time of one second.

**B. Plant model construction**

Maximum construction times of explicit-state and constraint-based plant models were 4.7 and 3.1 minutes respectively. The numbers of states in the explicit-state model and constraints in the constraint-based model are shown in Table II. The diversity of these numbers indicates that models of various complexities are included in the case study. A more interesting property of plant models is the percentage of supported transitions (also shown in Table II), which are more reliable than unsupported ones. This metrics characterizes the sufficiency of traces and the adequacy of plant parameter discretization given the traces. From the data we can see that the case study may be further improved, but, on the other hand, we are able to investigate plant models with the majority of transitions created according to our heuristic completion procedure.

**C. Closed-loop model checking**

Table II shows the results of closed-loop verification with generated plant models: the numbers of satisfied requirements (out of the maximum possible numbers) are presented together with model checking time. Each model checking run was limited to 48 hours. As visible from the table, verification times are smaller for constraint-based models. Still, there are subsystems for which verification even with constraint-based models is infeasible in realistic time. The supposed reason for this is the complexity of controller models.

There are cases of requirement violations. On one hand, violations (which are limited to the case of plant-and-controller requirements) may be explained by the wrong understanding of the generic PWR model by the authors. On the other hand, a smaller, simpler case study may be beneficial to analyze the relation of such violations with plant model reliability.

**D. Example**

Below, we focus on S7, a rather simple subsystem compared to others. Responsible for pressurizer pressure control, S7 controls spray valves and the pressurizer heater (9 real-valued plant model inputs) given the pressure $p$ and the liquid level $l$ in the pressurizer (2 real-valued plant model outputs). When $p$ is too high, closing signals for pressurizer spray valves are generated. Then, the pressurizer heater power depends on $l$, and the pressurizer heater is turned off if $l$ is too low.

All controller-only requirements, which related controller’s inputs and outputs, were satisfied in verification with both plant models. However, such a result is not surprising since they were also satisfied in open-loop model checking. A more interesting situation was observed for plant-and-controller requirements given in Section IV-A. Despite S7 being responsible for pressurizer pressure control, none of the requirements were satisfied for the constraint-based model and only requirements of the second type were satisfied for the explicit-state model. By examining the traces, we were able to see that such requirements were already violated in them, i.e. the generic PWR model may show behaviors violating the requirements, and the constructed constraint-based model is able to point this. Then, the examination of the explicit-state model revealed that some of the requirements were satisfied due to the inability of the model to leave some of its states given the existing controller – paths leading to such states were excluded from consideration due to fairness constraints.

**V. Discussion and Conclusions**

We have presented two methods of plant model synthesis (the explicit-state and the constraint-based ones) for closed-loop model checking and evaluated them on a case study involving an NPP simulation model. Both methods generalize simulation runs of the closed-loop system: while the explicit-state method treats occurrences of all plant model parameters as separate states, the constraint-based method infers dependencies between limited subsets of these parameters. According to Table II, the constraint-based method is superior over the explicit-state one in terms of time required for model checking. On the other hand, tight integration of MODCHK with NuSMV prevented us from exploring the case of explicit-state model checking (e.g. using the verifier SPIN) – it may potentially speed up verification of explicit-state models due to the small number of states in them.

Compared to the previous method of the authors [6], the proposed methods are to a large extent more scalable. While the method [6] generated models with up to 80 states from traces with up to 1000 elements in total, the proposed explicit-state method was able to handle 720000 trace elements and construct plant models with up to 4906 states. Supporting the same input data complexity, the constraint-based method constructs more concise models with more states. On the other hand, the proposed methods do not support LTL properties for the plant model as input data, but such properties may be difficult to obtain in practical cases, while the demand to construct larger models from larger trace sets does exist.
TABLE II
PROPERTIES OF GENERATED MODELS AND RESULTS OF CLOSED-LOOP MODEL CHECKING. TIME LIMIT IS INDICATED AS “TL”

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit-state method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of states</td>
<td>14</td>
<td>1355</td>
<td>1206</td>
<td>192</td>
<td>4906</td>
<td>1904</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>Fraction of supported transitions</td>
<td>0.754</td>
<td>0.062</td>
<td>0.025</td>
<td>0.411</td>
<td>0.360</td>
<td>0.303</td>
<td>0.598</td>
<td>0.714</td>
</tr>
<tr>
<td>Model checking time (min)</td>
<td>387.2</td>
<td>TL</td>
<td>278.6</td>
<td>TL</td>
<td>687.1</td>
<td>TL</td>
<td>2.1</td>
<td>1.0</td>
</tr>
<tr>
<td>Requirements satisfied</td>
<td>5 / 9</td>
<td>– / 30</td>
<td>38 / 38</td>
<td>– / 17</td>
<td>16 / 16</td>
<td>– / 30</td>
<td>18 / 21</td>
<td>8 / 20</td>
</tr>
<tr>
<td>Constraint-based method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of constraints</td>
<td>36</td>
<td>576</td>
<td>1275</td>
<td>144</td>
<td>357</td>
<td>234</td>
<td>26</td>
<td>20</td>
</tr>
<tr>
<td>Model checking time (min)</td>
<td>0.3</td>
<td>TL</td>
<td>278.6</td>
<td>616.3</td>
<td>0.1</td>
<td>TL</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Our approach has a number of limitations. First, correctness of the simulation model, which is usually constructed manually and hence is also influenced by the human factor, is not addressed. Then, generated models do not exhaustively represent possible plant behaviors since the input set of simulation traces is incomplete. On the other hand, no model can guarantee exhaustive analysis, and finding the trade-off between the richness of the model and the feasibility of verification is inevitable.

Considering the mentioned limitations, the issue of reliability of generated models is still important—otherwise, the methods will not be adopted in industrial practice. In Section IV-D, we have found that fairness constraints may cause some requirements to be satisfied while they must be violated. The issue of model reliability is also connected with the preparation of data for the methods. To address it, future work may involve adjustment of fairness constraints, new case studies, evaluation of the sufficiency of available traces, means to record them in order to maximize their utility in case studies, evaluation of the sufficiency of available traces, work may involve adjustment of fairness constraints, new case studies, evaluation of the sufficiency of available traces, means to record them in order to maximize their utility in terms of model quality, explorations of strategies of parameter discretization, and selection of proper cycle duration.

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) and by the Ministry of Education and Science of the Russian Federation, project RFMEFI58716X0032.

REFERENCES