Plant Model Inference for Closed-Loop Verification of Control Systems: Initial Explorations

Igor Buzhinsky, Valeriy Vyatkin

Citation:

Buzhinsky I., Vyatkin V. Plant Model Inference for Closed-Loop Verification of Control Systems: Initial Explorations. In Proceedings of the 2016 IEEE International Conference on Industrial Informatics (INDIN 2016), Poitiers, France, July 18–21, 2016, pp. 736–739

DOI: https://doi.org/10.1109/INDIN.2016.7819256

Plant Model Inference for Closed-Loop Verification of Control Systems: Initial Explorations

Igor Buzhinsky[1, 2] and Valeriy Vyatkin[1, 2, 3]
[1] Department of Electrical Engineering and Automation, Aalto University, Finland
[2] Computer Technologies Laboratory, 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*—Closed-loop model checking, a formal verification technique for industrial automation systems, increases the richness of specifications to be checked and often helps to reduce size of the state space to be verified compared with the open-loop case. To be applied, it needs two components – the controller and the plant models – to be coupled. While there are approaches for obtaining controller models from implementation, specification or behavior examples, little has been done regarding automation of plant model construction. This paper aims to solve the problem of automatic plant model construction from existing specification, which is represented in the form of plant behavior examples and temporal properties.**

I. Introduction

Formal verification and specifically model checking [1] is one of the ways to ensure dependability of industrial automation systems. Closed-loop model checking of control systems [2]–[4] enables proving not only controller properties but also properties of the plant under control, which are usually of more interest in the automation systems context. For that, the engineer is required to supplement the controller model with the corresponding plant model. The inclusion of this model into the system not only allows specifying properties to be verified in terms of the plant, which is natural in control engineering, but also often reduces the state space of the entire system compared to the open-loop controller model. This is achieved by omitting states unreachable in closed loop.

While manual work can be minimized during controller model preparation [5]–[7] even when its implementation is not entirely available, plant model development is an extra effort, which is not automated. Still, there is usually much information available to synthesize this model, such as plant interface, system specification and behavior examples. It can also be possible that the plant model is available, but it is continuous, which impedes its direct use in formal verification.

This paper presents a method to synthesize discrete plant models using behavior examples and temporal properties represented in linear temporal logic (LTL). Models are constructed as nondeterministic Moore automata and can further be translated into more convenient forms suitable for existing verifiers. As the means for plant model construction, we apply the translation-to-SAT approach [8], which assumes representing the synthesis problem as an instance of the Boolean satisfiability problem (SAT), running a third-party SAT solver for the generated Boolean formula, and reconstruction of the solution from the Boolean variable assignment produced by the solver. This approach is appealing due to the recent increase of SAT solver performance, which continues to improve over time.

The paper is structured as follows. In Section II, we briefly review related work. Next, in Section III, the proposed plant model inference method is described. It is evaluated in Section IV on two plant construction problems. The paper is concluded in Section V. The extended version of this paper will be submitted to IEEE Transactions on Industrial Informatics.

II. Related Research

A. Plant models in industrial automation

Testing and verification are the main means to ensure the correctness of industrial automation software. Both can be done either in *open loop* or in *closed loop*. In the latter, the control software is checked in an environment which resembles the devices with which the controller is supposed to operate in production. Modern simulation software tools, such as Simulink, Modelica, Apros and Ptolemy II, allow modeling of complex physical objects and processes. If one wishes to use the plant model in formal verification, it needs to be formal and often discrete. Finite-state machines, Petri nets [9], net condition/event systems (NCES) [10] and timed automata are among such models.

In [4], a methodology for semi-automatic plant model generation is proposed. It is suggested to start from a three-dimensional CAD drawing of the plant, then transform it into a simulation model (this is a software-specific task), and finally transform the simulation model into a formal one.

Another work [11] is devoted to constructing plant simulation models using information contained in PLC programs, such as the source code of the control application. While this is convenient (there is no need to prepare additional specification), such an approach can result only in very basic plant models. Still, this approach has some applicability in verification by simulation, but limitations are understandable as the model generated this way lacks dependencies between plant subcomponents.

B. Approaches to deterministic automaton synthesis

There is a large volume of publications dealing with the issue of constructing software models in the form of deter-

ministic finite automata (DFA), finite-state machines (FSMs) and extended finite-state machines (EFSMs). Some of these approaches might serve as a basis for the method to solve the problem considered in this paper.

One of the approaches [12] identifies DFA from sets of positive and negative behavior examples. Such automata are only able to distinguish incorrect software behaviors from correct ones. Richer models of EFSMs are considered in [13]. Both approaches, [12] and [13], translate the problem of model synthesis to an instance of the SAT problem: the solution of the problem is represented as a set of Boolean variables whose values are constrained to make them represent an automaton which complies with the given specification. To solve these instances, third-party SAT solvers are applied, which have recently become very efficient.

Alternative approaches to software model inference are also known, such as the method of constructing FSMs from traces and LTL properties [14] based on state merging [12]; the method of deriving EFSMs from traces with real-valued data [15], which is based on state merging and machine learning; and the approaches exploiting metaheuristic algorithms [7], [16], the former of which supports both traces and temporal properties as input specification. Temporal formulae as the only type of specification are considered in the problem of LTL synthesis [6], [17], [18].

III. Proposed Method

This section states the solved problem formally and outlines the proposed plant model synthesis method. The source code of the method, which is implemented in Java, is available online[1] as a module inside the EFSM construction project [13].

A. Plant model representation

We will represent the plant model to be constructed using the formalism of finite automata, or finite-state machines, which is widely applied in formal verification and software engineering in general. Moore-type automata, which incorporate outputs in their states, are most suitable for us, since this mapping of states into outputs clearly represents the usual situation when the plant's state is partially visible to the controller via its sensors. Next, plant models can be thought to be nondeterministic: this accounts for possible human intervention into the plant operation as well as allows not to model physical phenomena precisely. Hence, synthesized automata will be allowed to be potentially nondeterministic.

Other options for a formalism to represent discrete plant models include Petri nets, NCES and timed automata. While the first two options are more suitable for representing distributed plants, the third option captures continuous time aspects (timed automata are defined in terms of continuous time, but it can still be processed as a finite set of intervals). Finite automata are chosen instead since they are more basic, widespread, and can be converted into other formalisms without loss of information.

[1]https://github.com/ulyantsev/EFSM-tools/tree/plant-model-construction

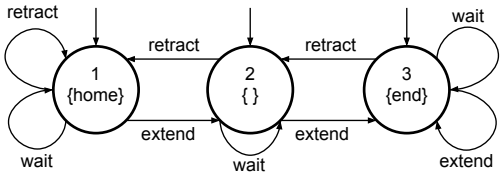

Fig. 1. Automaton representing the model of the pneumatic cylinder

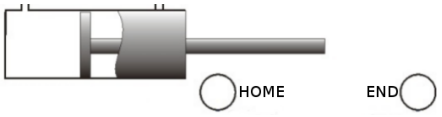

Fig. 2. Pneumatic cylinder

Formally, a nondeterministic Moore automaton is a sextuple $(S, S_0, I, O, T, \lambda)$, where S is the finite set of *states*, $S_0 \subset S$ is the non-empty set of *initial states*, I and O are the finite sets of *inputs* and *outputs* respectively, $T \in S \times I \times S$ is the *transition relation*, and $\lambda : S \to 2^O$ is the *output function*. Inputs are regarded in a generic sense: they can embody any discrete information from the controller, including events and variable values. As for outputs, the chosen form of the output function suggests that they should mainly be regarded as Boolean variables. Nevertheless, such variables are sufficient to represent any discrete value as well. The automaton must also satisfy the *completeness* requirement: for each input and state there is at least one transition to some other state.

The interaction of the plant and the control software (controller) is assumed to be cycle-based, but time intervals between cycles are irrelevant and are not obliged to be constant in particular. In the beginning of the interaction, the plant nondeterministically selects its initial state from the set S_0. Then, on each cycle, the controller reads the plant's output and chooses an input, after which the plant nondeterministically chooses a transition matching its current state and the provided input, and changes its state.

In Fig. 1 we introduce the example of an automaton, which will also be used later. This automaton represents the model of a pneumatic cylinder (see Fig. 2), which has three inputs (`extend`, `retract`, `wait`) and two outputs (`home`, `end`) representing three positions (home, end and intermediate). All the states of this model are initial, which is indicated by incoming arrows without source states. Multiple initial states is the only source of nondeterminism in this model.

B. Specification representation and problem statement

The first considered type of specification is a finite set of *traces*, or behavior examples. A trace of length l is a directed path $v_1 e_1 ... e_l v_{l+1}$, where each node is represented by plant outputs ($v_j \in 2^O, 1 \le j \le l+1$), and each edge is represented by a plant input ($e_j \in I, 1 \le j \le l$). The synthesized automaton is required to have each trace in its transition graph starting from an initial state. An example of a trace for the

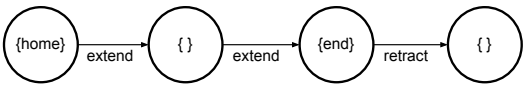

Fig. 3. Trace example for the pneumatic cylinder

cylinder example is shown in Fig. 3.

The remaining type of specification is a finite set of *LTL formulae*, or *LTL properties*, which the synthesized model is obliged to satisfy. LTL properties contain usual Boolean operators ($\wedge$, $\vee$, $\neg$, $\rightarrow$) as well as temporal ones (X, F, G, U). The following types of atomic propositions can be used in the formulae:

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

Since the plant-controller interaction is initiated by the plant, which selects an initial state with the first produced subset of outputs, on this first step neither of the `input` propositions are satisfied. Below are some examples of LTL properties for the cylinder model synthesis problem:

- $\mathrm{G}(\neg(\texttt{output}(\texttt{home}) \wedge \texttt{output}(\texttt{end})))$: "the cylinder cannot report being in `home` and `end` positions simultaneoulsy";
- $\mathrm{G}(\mathrm{G}(\texttt{input}(\texttt{extend})) \rightarrow \mathrm{F}\ \mathrm{G}(\texttt{output}(\texttt{end})))$: "if the cylinder receives the extension request eternally, then in the future it will start reporting being in the `end` position eternally".

Finally, the problem solved in this paper is to construct a nondeterministic Moore automaton, such that: the number of its states $|S|$ is fixed; each supplied trace is a valid example of its behavior; it complies with all given LTL properties and satisfies the completeness requirement.

C. Trivial solution for the case of traces only

If we assume that the specification is represented only by traces, then the considered problem can be solved trivially. To find the minimum automaton consistent with the traces, one needs to merge all trace nodes with identical outputs. When nodes are merged, their incoming and outgoing transition sets are also merged, and the resulting state is initial if and only if there was a root among the merged trace nodes. To satisfy the completeness requirement, missing transitions can be added, for instance, as self-loops. This procedure is linear of the total length of traces.

D. General scheme of the method

From now on, assume that the LTL specification is non-empty. We decided to base the plant model inference method on one of the existing methods for software model inference, which were briefly reviewed in Section II-B. Our approach aims at constructing nondeterministic Moore machines with a SAT-based approach, using the work [13] as the basis, since such an approach would be able to not only find the solution, but also prove that one does not exist.

Traces are encoded into Boolean formulae directly, but LTL properties are not: instead, when the constructed model violates them, negative traces are added to the specification and encoded into new Boolean constraints. The idea of using counterexamples was adopted from the work [14]. Instead of restarting the solver with the new, longer formula, we exploit the benefits of incremental SAT solving: only new constraints are passed to the solver, and it is requested to produce a new solution. This process ends when the identified model is finally correct. Briefly speaking, the stages of the proposed method are as follows:

1) translate the model inference problem instance, excluding LTL property compliance, into a Boolean formula;
2) run the SAT solver in the incremental mode;
3) if the solver spotted the absence of the solution, then either the specification is unrealizable or the supplied number of states is insufficient;
4) otherwise, reconstruct the solution from the found Boolean assignment;
5) if the solution violates LTL properties, generate counterexample traces, encode them into new constraints, pass them to the solver, request it to find a new solution and return to stage 3;
6) otherwise, apply post-processing of the obtained automaton.

As a SAT solver, *cryptominisat*[2] was chosen. The description of the translation to SAT is omitted for the sake of brevity.

IV. Experimental Evaluation

The introduced technique was applied in two case studies. More comprehensive evaluation, including experiments with randomly generated instances, will be considered in the following extended journal version of this paper.

A. Pneumatic cylinder

The first case study dealt with the discrete model of a pneumatic cylinder, which has been previously shown in Fig. 2. Such cylinders are common, for instance, for pick-and-place robots like the one considered in [19]. Normally, such a cylinder would be just a component of a more complex plant.

We wished to construct the model which would be able to report three positions – home, end and the intermediate one, and to switch positions based on controller's commands to extend or retract. To do this, we prepared three traces and six LTL properties. The examples of traces and LTL properties have already been given in Section III-B.

The resulting automaton is exactly the one which has been shown in Fig. 1. The method processed this example with the iteration number of three and the execution time of less than a second. Despite the successful result, the cylinder example is only suitable for the purposes of illustration and smoke testing of the method. A more comprehensive plant is considered in the following subsection.

[2]http://www.msoos.org/cryptominisat4/

B. Nuclear power plant model

For the second case study we used a generic nuclear power plant model provided by Fortum Power and Heat Oy, a power utility with nuclear power plant operation license in Finland. This model, implemented in the Apros modeling environment, includes the most important process, mechatronic and control components of a nuclear plant. It consists of a number of subsystems, among which we focused on the one which is responsible for activating emergency pumps. Since the entire nuclear plant is too complex, we decided to synthesise a discrete plant model only for several of its parameters – the ones directly related with this subsystem.

To prepare the data for plant construction, we executed ten simulations in Apros with the total length of 88 minutes (measured in simulated time; in reality it took around 26 minutes to record them). Inputs and outputs were measured each simulated second, resulting in the total trace length of 5289. Each output parameter was discretized into several levels based on the thresholds found in the considered control network. The prepared problem instance had 3 inputs (for each combination of pump states found in the traces) and 11 outputs (several outputs for each parameter). As temporal properties, we specified the validity of outputs (i.e. if a parameter has several discrete levels, then exactly one of them must be generated on each step) and the smoothness of their changes (a discrete level can only change to a contiguous level).

We executed the plant model construction method with various numbers of states, and found that the value of 12 is the minimum possible one. For this number of states, the execution time of the method was 12.3 seconds and the number of iterations was 15. The constructed plant model was then coupled with the NuSMV model of the nuclear power plant control subsystem, and several simple temporal properties were verified.

V. Conclusion

In this paper, a method of discrete plant model construction was proposed. We believe that this method is applicable to aid closed-loop model checking. Still, there are several issues which need to be discussed. They are mentioned briefly here and will be discussed in detail in the extended version of the paper. First, the method delivers possibly imprecise models which still can be useful in model checking since they can find faults in the verified system, although possibly not all of them. Next, the method is not addressing yet complex, modular plants. Our last concern is the source of input specification. Ideally, traces should be obtained based on a more complex, often continuous simulation model (e.g. a hybrid automaton, a set of differential equations, an Apros model). If there are no such models, other automatic behavior generation strategies, which depend on the plant, might also be available.

Future work may include the evaluation of the method on larger plants and more extensive application of the produced plant models in model checking.

Acknowledgments

This work was financially supported, in part, by the SAUNA project (funded by the Finnish Nuclear Waste Management Fund VYR as a part of research program SAFIR2018) and by the Government of Russian Federation, Grant 074-U01.

We thank Antti Pakonen from VTT Technical Research Centre of Finland for preparing the NuSMV model of the nuclear power plant subsystem used in experimental evaluation.

References

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

[2] S. Preuße, H. Lapp, and H. Hanisch, "Closed-loop system modeling, validation, and verification," in *17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2012, pp. 1–8.

[3] C. Gerber, S. Preuße, and H.-M. Hanisch, "A complete framework for controller verification in manufacturing," in *15th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2010, pp. 1–9.

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

[5] C. Pang and V. Vyatkin, "Automatic model generation of IEC 61499 function block using net condition/event systems," in *6th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 2008, pp. 1133–1138.

[6] C.-H. Cheng, C.-H. Huang, H. Ruess, and S. Stattelmann, "G4LTL-ST: Automatic generation of PLC programs," in *Computer Aided Verification*. Springer, 2014, pp. 541–549.

[7] D. Chivilikhin, V. Ulyantsev, and A. Shalyto, "Combining exact and metaheuristic techniques for learning extended finite-state machines from test scenarios and temporal properties," in *Proceedings of the 13th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, Dec 2014, pp. 350–355.

[8] A. Biere, M. Heule, and H. van Maaren, Eds., *Handbook of satisfiability*. IOS press, 2009.

[9] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[10] M. Rausch and H.-M. Hanisch, "Net condition/event systems with multiple condition outputs," in *INRIA/IEEE Symposium on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 1995, pp. 592–600.

[11] 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.

[12] M. J. Heule and S. Verwer, "Software model synthesis using satisfiability solvers," *Empirical Software Engineering, Springer*, vol. 18, no. 4, pp. 825–856, 2013.

[13] V. Ulyantsev and F. Tsarev, "Extended finite-state machine induction using SAT-solver," in *Proceedings of the 14th IFAC Symposium "Information Control Problems in Manufacturing (INCOM)"*. IFAC, 2012, pp. 512–517.

[14] N. Walkinshaw and K. Bogdanov, "Inferring finite-state models with temporal constraints," in *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Computer Society, 2008, pp. 248–257.

[15] N. Walkinshaw, R. Taylor, and J. Derrick, "Inferring extended finite state machine models from software executions," *Empirical Software Engineering, Springer*, vol. 21, no. 3, pp. 811–853, 2016.

[16] D. Chivilikhin and V. Ulyantsev, "MuACOsm: a new mutation-based ant colony optimization algorithm for learning finite-state machines," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. ACM, 2013, pp. 511–518.

[17] B. Jobstmann and R. Bloem, "Optimizations for LTL synthesis," in *Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2006, pp. 117–124.

[18] R. Ehlers, "Symbolic bounded synthesis," *Formal Methods in System Design, Springer*, vol. 40, no. 2, pp. 232–262, 2012.

[19] S. Patil, V. Vyatkin, and M. Sorouri, "Formal verification of intelligent mechatronic systems with decentralized control logic," in *17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2012, pp. 1–7.