Closed-loop verification of a compensating group drive model using synthesized formal plant model

Polina Ovsiannikova, Daniil Chivilikhin, Vladimir Ulyantsev, Anatoly Shalyto
Computer Technologies Laboratory, ITMO University, Saint Petersburg, Russia
Email: polina.ovsyannikova@corp.ifmo.ru, chivdan@rain.ifmo.ru, ulyantsev@rain.ifmo.ru, shalyto@mail.ifmo.ru

***Abstract*—Cyber-physical systems are spread among multiple domains of human activity. Their behavior should be strictly validated as most of them are critical. One perspective direction of ensuring system correctness is verification using model checking. A new method for closed-loop model checking of cyber-physical systems has recently been introduced that involves automatic formal plant model synthesis followed by temporal properties verification. The formal model is inferred based on traces gathered from the simulation model. Then, verification of temporal properties for the whole closed-loop system can be performed using the NuSMV tool. The main purpose of this paper is to investigate the applicability of the aforementioned method by performing a case study on the example of a compensating group drive simulation model provided by our industrial partners.**

I. Introduction

There is hardly any such scope where no cyber-physical systems (CPS) are applied as they have become an inalienable part of the modern world. In many cases they play a critical role and their behavior is required to be correct. There are several approaches for checking whether the developed CPS follows its specification, among which are testing and formal verification. Standard testing does not consider all the range of possible system states, so it cannot prove that the system is correct. Moreover, in some domains CPS with wrong behavior can make irreparable harm to the environment or people, so the better alternative for ensuring system correctness is formal verification. In particular, in the presented paper we will discuss the model checking method [1], [2] for formal verification.

Before using model checking for CPS verification the desired strategy of this methodology should be chosen. There are two main approaches for model checking: open-loop and closed-loop [3], [4] verification. With regard to the first approach, the plant model is not taken into consideration, but only the controller model is tested. This leads to the fact that the controller may have any inputs including ones that the plant can never produce leading to the state explosion problem. However, this approach can be useful in systems of limited complexity.

In the second approach, both plant and controller formal models are required. Having a formal plant model, plant output stubs in the controller can be replaced by actual model outputs. Thereby, the loop between the controller and the plant is closed. Unlike the open-loop strategy, here the problem of state explosion hardly occurs as the state space is reduced. Therefore, in most cases, even complicated systems can be verified.

978-1-5090-6505-9/17/$31.00 2017 IEEE

In this paper closed-loop model checking will be discussed. To make this method applicable, first the formal CPS model should be available. A formal model normally consists of two components – formal model of the plant itself (*plant model*), that is independent and can act in any way allowed by its physical constraints, and formal controller model (*controller*) that receives output signals from the plant model and generates inputs to the plant model based on them. The problem is that the plant model is almost never provided due to various reasons such as developing it separately from the system itself and then necessity to maintain it with every change of the real system. Generally, resources spent on this kind of work may never be paid off. Hence, there should be a technique for formal plant model inference. For example, [5] considers creating modular formal models from elementary automata. In [6], [7] methods for obtaining formal plant model from 3D CAD models are proposed. The aforementioned methods have such disadvantages as the requirement that the simulation model must be in a specific format and semiautomatic formal model creation.

In [8], [9] and [10] methods for automatic formal model inference using simulation model execution traces have been introduced. They do not have restrictions for the source simulation model format and do not require human assistance during formal model creation. As they use traces for model generation, it is necessary to select such traces so that the resulting formal model will be precise enough. Research in this area was performed in [11], where an approach of gathering traces for inferring models satisfying this condition was suggested.

The formal model can be inferred using two main approaches [8], [9], [10]. One of them is the *explicit-state* method. It implies creating an automaton with states produced from outputs and transitions made of inputs between consequent outputs encountered in traces. Since sometimes traces do not cover the whole system behavior, the first issue is deadlocks caused by so-called unsupported transitions. To resolve it, such transitions should be created in advance. All this leads to the approximately unrealistic implementation to the real world. This is due to the fact that to make the system adequate all possible system states need to be covered by traces and, consequently, by automaton states, which can and

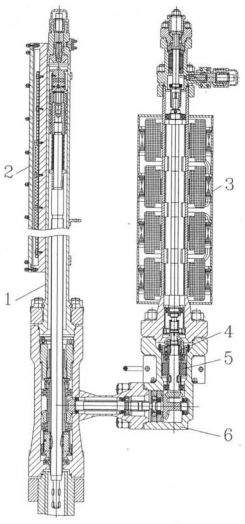

Fig. 1. Compensating group. 1 – screw mechanism, 2 – reference point sensor, 3 – stepper motor, 4 – overrunning clutch, 5 – displacement sensor, 6 – reducer. Image from http://shkolnie.ru/biolog/18152/index.html

will lead to the state explosion problem in complex systems.

The other approach is *constraint-based*. Its core is that every plant model output and input has its corresponding state variable. Continuous variables are assumed to be discretized, therefore the amount of possible values should be equal to their discrete intervals count. The model is inferred in the form of simple constraints on the consequent values of variables.

The purpose of this paper is to perform a case study on the example of a compensating group drive model using the method for automatic formal plant model synthesis introduced in [8], [9].

II. Case study: nuclear reactor compensating group drive model

A. Nuclear reactor compensating group

The method proposed in [8], [9] is studied using an example of a drive of the compensating group (CG) which is a part of a shipboard nuclear reactor system – a set of equipment and subsystems designed to convert the energy of nuclear fission into thermal energy providing electricity and mechanical energy for ship movement. The structure of the shipboard system includes subsystems and elements that ensure normal operation of the reactor and its safety together with the systems of the nuclear power plant.

CG is used for slowing down and terminating or speeding up the nuclear reaction. The drive of CG moves compensating rods into the active zone of the reactor. It consists of a screw mechanism, a reducer, a stepper motor, a displacement sensor and a sensor of reference points (Fig. 1). Vertical displacements of the screw and the CG connected with it are performed when the ball nut is turned by the stepper motor.

B. Simulation model of the CG drive

We have received the CG drive simulation model (Fig. 2) from our industrial partners. It consists of a CG drive, controller and external control module that can be considered as a human-machine interface. The model of the plant itself (CG drive) is viewed as a black box. This model was implemented using MatLab/Simulink software [12].

The system works as follows. The controller receives plant outputs and, taking them into account, generates inputs for the plant. The output that is produced by the model is the current position of the CG measured in millimeters (mm) with a maximum at 1160 mm and a minimum at -5 mm with 1 mm acceptable error. After receiving the current position of the CG from the plant and the desired target position of the CG, the controller generates the movement `Direction`. Possible values of the `Direction` variable are: -1 – move down, 0 – hold the current position, 1 – move up. There are also two emergency signals: `StopSignal` that prevents any movement of the CG and `MaxDownSignal` that makes the CG reach the lower bound with denial of any further commands until `MaxDown` is completed.

C. Simulation traces generation

To perform formal verification the formal model of the system is needed. For this purpose the method suggested in [8] was used where the formal plant model is inferred using simulation traces gathered from the given simulation model. In this case study each trace element is represented by a tuple (`Position`, `Direction`) as only these variables are responsible for the main plant functionality.

As the given model is simple enough and all possible system states can be easily observed, the traces gathering strategy was created intuitively as follows. The controller was configured to lift the CG up by 5 mm and stop it. This sequence repeated until the CG reached the upper bound. After the peak, the CG was still moving up for some time and then it started moving down with the same logic until reaching the lower bound. Thus, resulting traces included every key position combined with all possible movement directions.

D. Formal plant model construction

To generate the formal plant model, first it is necessary to discretize continuous variables. In the presented case study such variable is `Position`. In order to check Linear Temporal Logic (LTL) properties (will be discussed in Section II-F) that refer to the next or current value of the `Position` variable, the whole range of possible values $[-5; 1160]$ was split into intervals which represent control points and their ambient region and interspace between them. For example, for $[-5, 5]$ the descretized intervals can be depicted as in Fig. 3.

We chose the constraint-based plant model inference method to be studied. Traces and configuration files were sent to the formal model construction tool and as a result the plant model in NuSMV [13] format was inferred.

After model construction it is necessary to ensure that it is precise enough. For this reason the formal model conformance checking method from [11] was used. Randomly we generated upper and lower boundaries that the CG was supposed to reach. Also, arbitrary reference points were defined, where

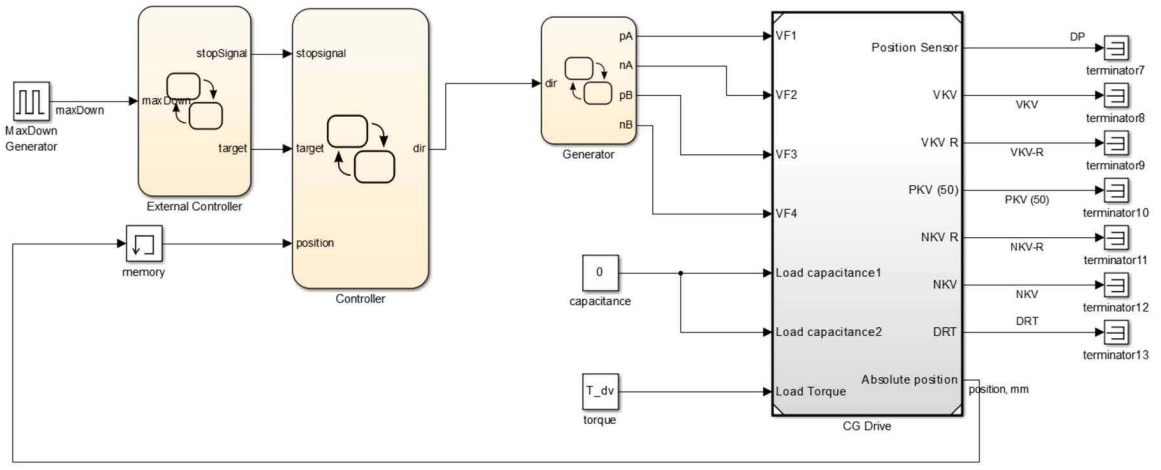

Fig. 2. Matlab/Simulink compensating group simulation model

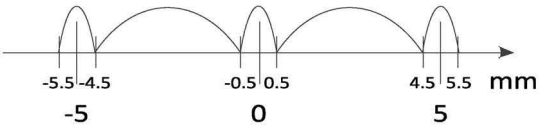

Fig. 3. An example of discretizing interval $[-5; 5]$

the compensating group stopped at random time. The new trace files describing this behavior were generated and sent to the conformance checking tool. As a result, it created Computation Tree Logic (CTL) properties for each trace and launched NuSMV to perform model checking. The average percentage of traces accepted by the formal plant model was 99.95%, which means that the model is sufficiently adequate to the simulation model and can be used for verification.

E. Controller model construction

In the given simulation model the controller was implemented using Simulink Chart Block (Fig. 4). As, to the best of our knowledge, MatLab does not have functionality that allows to convert blocks of this type into NuSMV code, the formal model of the controller was created manually and then composed with the plant formal model constructed on the previous step.

F. Formal verification

LTL is a language for describing linear-time system properties with atomic expressions (elementary statements about the system) and *temporal operators*, e.g.:

- $\varphi_1 \, \mathbf{U} \, \varphi_2$ – `Until` – φ_1 must be true until φ_2 is true. After that φ_1 is not required.
- $\mathbf{G} \, \varphi$ – `Globally` – φ must be true for the whole range of system paths.
- $\mathbf{X} \, \varphi$ – `Next` – φ must be true for the next state.

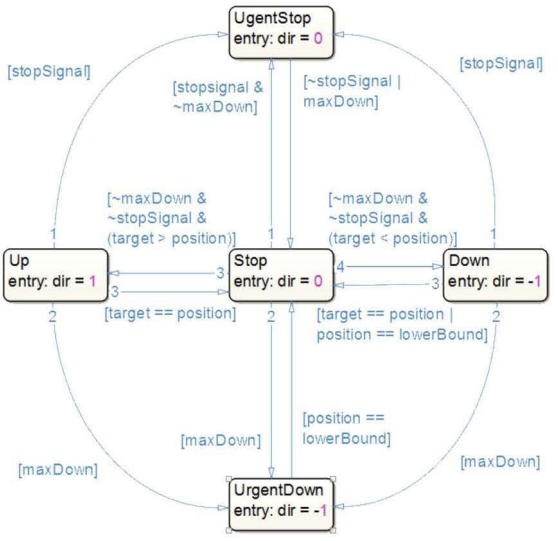

Fig. 4. State diagram of the controller

Four LTL properties described in Table I were verified using NuSMV. The whole system model with LTL properties was sent to NuSMV and the following results were obtained.

The first property is satisfied. It was important to check if `StopSignal` has the second priority as `MaxDown` signal is the strongest emergency signal in the studied system. The second property is also satisfied. CG will not accept any commands until `MaxDown` is completed.

The third and fourth properties are violated, it happens due to several points. First, in traces `Position` is measured with a precision of 0.0001 mm. Second, discretized intervals cannot be smaller than 1 mm in length. A stepper motor that moves the CG has a fixed step, so position error can be calculated as a half of a motor step. Hence, the exact position of the CG is unknown, however we can determine whether it belongs to the allowed range. After the plant receives the signal to move

TABLE I
TEMPORAL PROPERTIES FOR THE CG SYSTEM

Name	Temporal property	Comment	Correct value	Obtained value
φ_1	$\forall$ p in [0..466] $\mathbf{G}(\texttt{StopSignal} \wedge \texttt{!MaxDown} \wedge (\texttt{Pos} = p) \rightarrow \mathbf{X}(\texttt{Pos}) = p)$	If the `StopSignal` was sent to the CG and no emergency `MaxDown` signal was detected then the CG will not move.	+	+
φ_2	$\mathbf{G}(\texttt{MaxDown} \rightarrow (\texttt{Dir} = -1)\,\mathbf{U}(\texttt{Pos} = -5))$	If the emergency signal to go down was sent to the CG, it will move down until reaching the lower bound. This is used for urgent termination of nuclear reaction.	+	+
φ_3	$\forall n \in [0..233]\,\mathbf{G}(\texttt{Dir} = -1 \wedge \texttt{Pos} = 2n+1 \rightarrow \mathbf{X}(\texttt{Pos} = 2n))$	If the CG is in a control point and the command was given to move down, then the next position of the CG will be inside the interval previous to the next control point.	+	−
φ_4	$\forall n \in [0..232]\,\mathbf{G}(\texttt{Dir} = 1 \wedge \texttt{Pos} = 2n+1 \rightarrow \mathbf{X}(\texttt{Pos} = 2n+2))$	If the CG is in a control point and the command was given to move up, then the next position of the CG will be inside the interval next to the next control point.	+	−

in some direction, the state cannot be changed immediately since first the CG needs to leave the previous interval and this takes some time. That is why the LTL properties that require immediate transition after up/down signals are violated.

III. CONCLUSION

In the presented case study the method for formal CPS verification was applied to a model of a real system that controls the position of a CG of a floating shipboard nuclear reactor. The formal plant model was built using the constraint-based approach [8], [9], [10]. The constructed formal plant model passed the traces coverage test with an average result of 99.95% which means that it is precise enough to be used for formal verification. Then, four LTL constraints were checked.

The most important and safety-critical system properties φ_1 and φ_2 were proven to be correct. During model checking we detected an issue with verifying immediate transitions between an interval that contains a control point and its allowed range and an interval between control points. Due to acceptable error of CG position measurement, we cannot claim that after a specific command the system will immediately move to the next state. However, such or similar properties still need to be checked. For this case the following solution can be proposed.

If the system requires time to move from one state to another then this time should be specified and taken into account during property checking. For example, it can be checked that after a certain number of steps the model will eventually move to the required state. Creating methods that would allow to infer such models is part of ongoing work.

ACKNOWLEDGEMENTS

This work was supported by the Ministry of Education and Science of the Russian Federation, project RFMEFI58716X0032.

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

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

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

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

[7] E. Lobo, J. Fertuzinhos, J. P. M. A. Silva, and J. Machado, "Obtaining plant models for formal verification tasks from 3D CAD models: Which is the best approach?" *Advanced Materials Research*, vol. 630, pp. 283–290, 2013.

[8] I. Buzhinsky and V. Vyatkin, "Automatic inference of finite-state plant models from traces and temporal properties," *IEEE Transactions on Industrial Informatics*, 2017.

[9] ——, "Plant model inference for closed-loop verification of control systems: Initial explorations," in *IEEE 14th International Conference on Industrial Informatics (INDIN)*, pp. 736–739.

[10] I. Buzhinsky, A. Sandru, A. Pakonen, D. Chivilikhin, V. Ulyantsev, A. Shalyto, and V. Vyatkin, "Generation of formal plant models based on simulation environments." [Online]. Available: http://eurosim2016.automaatioseura.fi/images/sas/posters/Eurosim2016_poster_Buzhinsky.pdf

[11] D. Avdyukhin, D. Chivilikhin, G. Korneev, V. Ulyantsev, and A. Shalyto, "Plant trace generation for formal plant model inference: methods and case study," in *15th IEEE International Conference on Industrial Informatics*, July 2017, pp. 746–752.

[12] Matlab Simulink homepage. [Online]. Available: https://www.mathworks.com/products/simulink.html

[13] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: A new symbolic model verifier," in *International conference on computer aided verification*. Springer, 1999, pp. 495–499.