Reconstruction of Function Block Controllers Based on Test Scenarios and Verification

Daniil Chivilikhin*, Ilya Ivanov* Anatoly Shalyto*, and Valeriy Vyatkin*†‡
*Computer Technologies Laboratory, ITMO University, Saint Petersburg, Russia
Email: chivdan@rain.ifmo.ru, shalyto@mail.ifmo.ru
†Department of Electrical Engineering and Automation, Aalto University, Finland
Email: vyatkin@ieee.org
‡Department of Computer Science, Computer and Space Engineering, Lulea Tekniska Universitet, Sweden

***Abstract*—The paper addresses the problem of reverse engineering a function block (FB) in situations when its source code is either not available or is too complex to understand. The proposed approach builds up on a recent method for reconstructing FBs based on testing and a search-based optimization algorithm. In our work the method is augmented with candidate solution verification using the NuSMV model checker. Verification is done in a closed-loop way using a manually constructed surrogate model of the plant and environment.**

I. Introduction

IEC 61499[1] is an international standard for industrial process measurement and control systems [1]. The specification of IEC 61499 defines a generic model for distributed control systems and is based on the IEC 61131 standard[2]. An application in IEC 61499 is represented as a network of function blocks (FB) which encapsulate both logic and data processing. Each FB has an interface that defines its input/output *events* and input/output *variables*. Function blocks can be either *basic* or *composite*. A basic function block is described in terms of an *execution control chart* (ECC), which is a special type of Moore finite-state machine (FSM) – every FSM state can have several associated *output actions*. Each output action references one or zero *algorithms* and one or zero events. Algorithms are used for changing output variable values and can be implemented as defined in compliant standards, but commonly the *Stuctured Text* language is used. A composite function block is represented with a network of interconnected blocks, basic and/or composite.

In this work we concentrate on basic function block *reconstruction* or reverse engineering. However all proposed algorithms may be applied to composite blocks with slight or no changes.

The FB reverse engineering problem arises when the initial source code of a function block has been lost and only the binary executable is available. An approach has recently been proposed [2] that is claimed to be able to reconstruct ECCs of basic function block logic using testing: the original FB is placed into a test environment, its behavior is recorded in the form of *execution scenarios* or traces and then a search-based optimization method (e.g. evolutionary algorithm [3]) is used to find an ECC that complies with these scenarios. The essence of the synthesis procedure in [2] is that candidate ECCs are evolved by applying small changes (mutations) and evaluated using a *fitness function* that measures the degree of compliance with given execution scenarios.

A serious and yet unresolved issue with this approach is the question of completeness of the test set. Since, without the availability of source code, there is no way to ensure that the test set covers all necessary behaviors of the original FB, there are no guarantees that the reconstructed FB will be (at least behaviorally) equivalent to the original one. Approaches such as model-based testing [4] may be useful for coping with this.

Our paper attempts to approach this problem from another side – using FB verification [5]–[7]. We augment the input data with temporal properties that the original FB satisfies. These are expressed in temporal logics (e.g Linear Temporal Logic – LTL) and can often be derived from the documentation of the original FB or even be available explicitly (especially for reliable systems such as power plant control applications). We can safely assume that these temporal properties cover the most important functionality of the FB. Therefore, if we can guarantee that the synthesized FB satisfies these properties, that the important functionality of the original FB will be retained.

Our approach is based on the FB synthesis method proposed by the authors of [2], but incorporates FB verification in the process. The contributions of this paper are the following.

1) We propose an approach to fast on-the-fly FB verification which lies in the middle between open-loop and closed-loop verification [8].
2) We propose an efficient method of combining testing and verification in the process of FB synthesis.

At the moment we limit ourselves to basic FBs that implement logic control, i.e. all input and output variables are Boolean. We should note that this limitation does not, however, make the problem easy – it still remains at least NP-hard.

The rest of the paper is structured as follows. In Section II we review background information on function blocks, verification, function block synthesis and formulate the addressed problem. We then describe the approach proposed in this paper in Section III. Section IV is dedicated to an experimental case

[1]https://webstore.iec.ch/searchform&q=61499
[2]https://webstore.iec.ch/searchform&q=61131

978-1-5090-2870-2/16/$31.00 ©2016 IEEE

study on which we evaluated the proposed method. Finally, Section V is devoted to discussion of the results and drawing conclusions.

II. Background and problem statement

A. Definitions and problem statement

In this section we include necessary definitions and provide a formal problem statement.

A basic function block is defined by an *interface* and a special Moore finite state machine called an *execution control chart* (ECC). The interface defines input/output events and input/output variables of the FB. An example of a FB interface is shown in Fig. 1a.

An ECC with Boolean variables can be defined as a nine-tuple $\langle E, I, Y, Z, O, y_0, \phi, \sigma, \eta \rangle$, where E is a set of input events, I is a set of Boolean input variables, Y is a set of states, Z is a set of output events, O is a set of Boolean output variables, y_0 is the initial state, $\phi : Y \times E \times \{0,1\}^{|I|} \to Y$ is the transition function, $\sigma : Y \to Z^*$ is the output actions function and $\eta : Y \to \{0,1,x\}^{|O|}$ is the output variables function, where 0 and 1 mean assigning a corresponding value to a variable and x means preserving its current value. An example of an ECC is given in Fig. 1b.

The linear temporal logic language consists of problem dependent propositional variables, Boolean logic operators $\vee, \wedge, \neg, \to$ and a set of temporal operators, for example:

- Globally – $G(f)$ means that than f has to hold for all states;
- Next – $X(f)$ means that f has to hold in the next state;
- Future – $F(f)$ means that f has to hold in some state in the future.

For example, if x_1 and x_2 are propositional variables, then $G(x_1 \to X(x_2))$ is a LTL formula meaning "Always: if x_1 then in the next state x_2".

An *execution scenario* is a sequence of four-tuples $\langle e, i, z, o \rangle$, where $e : E^*$ is an input event, $i : \{0,1\}^{|I|}$ is a vector of input variable values, $z : Z^*$ is an output event and $o : \{0,1\}^{|O|}$ is a vector of output variable values. An ECC is said to satisfy an execution scenario if when it is executed and given the input events and variables from the scenario it will produce corresponding output events and variables from the same tuple.

Let a be a set of LTL formulas that the original FB satisfies and b be a set of scenarios derived from testing the original FB. The goal is to find an ECC c which will satisfy the following constraints.

1) All formulas from a must hold for ECC c.
2) ECC c must satisfy all scenarios from b.

B. Overview of ECC synthesis from execution scenearios

Since our approach is based on [2], in this section we briefly review the ECC synthesis method from [2]. Their algorithm is based on metaheuristic search-based optimization [9]. In general, metaheuristics are used for finding good solutions of hard optimization problems (i.e. ones for which an exact solution algorithm is too computationally hard) in reasonable time.

The search process in [2] begins with generating random initial candidate solutions (random ECCs). Then, the search space, which is the set of all possible ECCs with a fixed interface, is explored using so-called *mutation operators*: procedures that make (small) changes to ECC structure. For example, the following operators are used in [2]:

- select random transition and change its destination state;
- select random transition and delete it;
- select random state and add a transition to another random state.

Each new candidate solution is evaluated using a *fitness function*: it measures the degree to which the solution complies with execution scenarios. This is done in the following way. Each scenario is processed separately. The ECC is fed with input tuples (input event and input variable values) from the scenario, and its outputs – output event and output vairable values – are recorded. Then the output tuples generated by the candidate ECC are compared to the output tuples from the scenario using Levenshtein's string edit distance [10]. Comparison results are normalized according to scenario length and averaged across all scenarios. The fitness function also measures some additional parameters: the position of the first error in output tuples (the greater – the better) and the number of state changes the ECC makes while processing scenarios (the smaller – the better).

The final fitness function from [2] has the form:

$$F = c_1 F_{ed} + c_2 F_{fe} + c_3 F_{sc}, \tag{1}$$

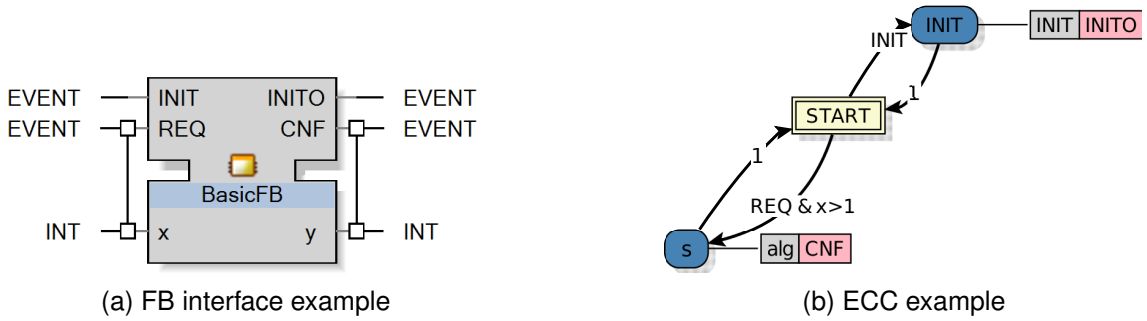
where F_{ed} is based on string edit distance, F_{fe} is based on the position of the first error the candidate ECC makes and F_{sc} is the number of state changes, and c_1–c_3 are constants. Constants are selected in such a way that all ECCs with a fitness value greater than 1.1 satisfy all scenarios and temporal properties. The optimization algorithms tries to maximize the fitness function.

C. Function block verification

An industrial application typically consists of a plant and a corresponding controller. In IEC 61499 both are represented using function blocks. Here we are concerned with controller verification.

There are two main approaches to controller-plant systems verification – open-loop and closed-loop [8]. In open-loop verification the controller is verified in an isolated manner. In contrast to this, closed-loop verification implies coupling of the controller and plant models and simultaneous verification. The latter approach is much more logical than the former, since we are interested in how the controller behaves while interacting with the plant and not by itself. Furthermore, in open-loop verification we might not be able to verify seem-to-be correct temporal properties since nothing constrains the inputs of the controller.

However, closed-loop verification has a serious issue – we need not only a formal model of a controller (which can



(a) FB interface example (b) ECC example

Fig. 1. Function block interface and execution control chart examples

be easily generated automatically), but also a formal model of the plant. Sometimes plant models can also be generated automatically [7], but this can result in very large models which will be hard to verify. For example, even with the plant abstraction technique [7], closed-loop verification of the Pick-and-Place manipulator system took 294 seconds. Since in our work we will have to verify a very large number of candidate solutions, we cannot afford such large verification times.

III. Proposed approach

To solve the stated problem and cope with closed-loop verification of candidate controllers, we propose the following method. Its base is the parallel search-based optimization algorithm from [2]. The algorithm is extended with new fitness function components and a mutation operator for handling temporal properties verification. Candidate solutions verification is based on a surrogate plant model, usage of which drastically decreases verification time. We use the NuSMV [11] model checker for FB verification.

A. Closed-loop verification with surrogate plant model

The approach presented in this paper lays in the middle between open- and closed-loop verification.

1) Controller model: The following approach similar to the one used in [7] is used to convert the controller (ECC) to an SMV model (Symbolic Model Verifier). We create a variable $eccState : Y$ to store the current state. For each input and output event we create a Boolean variable, indicating whether this event occurred or not. For each input and output variable we create a corresponding Boolean variable.

Initially, $eccState$ is set to the initial state of the ECC. Then, on each step its value is updated according to its current value, input events and input variable values. Output events and variables are calculated according to the new value of $eccState$. ECC output actions can include generation of an output event, setting some output variable(s) $true$, setting some output variable(s) $false$, or leaving the output variable(s) unchanged.

2) Surrogate plant model: Then we manually create a surrogate model of the plant and environment (e.g. user actions) and couple it with the controller model. All we require from the surrogate model is that to interact with the controller in a way the real environment and plant do. All other details are redundant. This means that the surrogate model can be much simpler than the full plant-environment model, which can (and does) reduce verification time dramatically. A disadvantage of the approach is that we need to find this surrogate model – this can also be a nontrivial task.

The SMV model of the plant and environment is finally concatenated with the controller model.

B. Verification-aware fitness function

Given a specification expressed using temporal formulas, NuSMV checks whether a given SMV model satisfies the specification. For each formula it either outputs that it is satisfied, or prints a *counterexample* as an evidence that the formula is not satisfied: a sequence of input/output event and input/output variable values. Counterexamples may be viewed as execution scenarios.

First, we introduce a new fitness function component $F^{\text{sat}}_{\text{smv}}$ – the ratio of the number of satisfied formulas to the total number of temporal formulas. Apart from that, we also use the counterexamples to the unsatisfied formulas in our algorithm. The use of counterexamples forces us to limit ourselves to Linear Temporal Logic, excluding pure Computational Tree Logic (CTL) formulas upfront. This is due to the fact that for unsatisfied CTL formulas that use the "Exists" CTL operator no counterexample can be derived – there is simply no evidence that some path in the SMV model does not exist. If, for instance, our algorithm comes to a state when all scenarios are satisfied but a CTL-formula with an existential operator is not satisfied, then we will not have any other information except that the formula is not satisfied. Thus, the search process will be stuck.

We also use an additional component $F^{\text{ce}}_{\text{smv}}$ that is based on the length of the longest counterexample $l_{\max}$ to the unsatisfied formulas. The idea behind this is that solutions that result in longer counterexamples to LTL formulas are probably better than solutions with short counterexamples. Since there is no maximum length of a counterexample, we use the following normalized function that increases with $l_{\max} > 0$:

$$F^{\text{ce}}_{\text{smv}} = \begin{cases} 1, & \text{if } l_{\max} = 0; \\ 1 - \frac{1}{(1+\frac{1}{10}l_{\max})^{\frac{1}{10}}}, & \text{otherwise.} \end{cases}$$

The value of $F^{\text{ce}}_{\text{smv}}$ equals 1 when $l_{\max} = 0$ and $F^{\text{ce}}_{\text{smv}} \underset{l_{\max}\to\infty}{\to} 1$, which is exactly what we need.

To sum up, our fitness function is based on (1) but has two additional components based on verification results:

$$F = c_1 F_{ed} + c_2 F_{fe} + c_3 F_{sc} + c_4 F^{\text{sat}}_{\text{smv}} + c_5 F^{\text{ce}}_{\text{smv}}.$$

An issue with using verification results in fitness evaluation is that though verification time is much smaller than when using a full plant model, it is still very considerable. For example, in our experiments with the Pick-and-Place manipulator [12], verification of one candidate solution takes about 0.1–0.2 seconds on a personal computer. This is up to an order of magnitude greater than the time needed to evaluate other fitness function components based on scenarios. To cope with this we use the following approach.

Instead of evaluating the verification-based fitness components for all solutions, we only do it with some small probability p ($p = 0.99$ in experiments). With probability $1-p$ we take the values of $F^{\text{sat}}_{\text{smv}}$ and $F^{\text{ce}}_{\text{smv}}$ of the parent of the current candidate solution as an estimation. Of course, this may lead to missing good solutions, but this is the price we have to pay to make the method feasible.

In addition, if $c_1 F_{ed} + c_2 F_{fe} + c_3 F_{sc}$ is greater than some threshold (we used 99% of the maximum value of these components) we evaluate verification-based fitness components. This allows us to avoid missing near-optimal solutions and also ensures that the final solution will be correct.

C. Verification-aware mutation operator

As was mentioned before, if a model does not satisfy a temporal formula, NuSMV provides us with a counterexample to the formula. A counterexample can be viewed as an execution scenario. In addition to using the counterexample for fitness evaluation, we use it to create a verification-aware mutation operator. This operator has the feature of increasing the probability of removing/changing transitions that are part of the counterexample and is similar to the one proposed in [13] for extended finite-state machine (EFSM) inference.

The operator associates each ECC transition t with its weight w_t. Initially, $w_t = 1$ for all transitions. Then we execute each counterexample and increase the weights of visited ECC transitions by λ ($\lambda = 1$ in the experiments). We then use the roulette wheel method [14] with probabilities proportional to the transition weights w_t for selecting one transition to modify by the mutation operator. The selected transition's destination state is changed to another state selected uniformly at random from the set of all ECC states.

IV. Experimental case study: synthesizing controllers for a Pick-and-Place manipulator

To evaluate our approach we performed an experimental case study on the Pick-and-Place (PnP) manipulator system [12] shown in Fig. 2. The PnP manipulator is a robotic hand: it has three input tracks (1, 2, 3) and one output track (V). It has two horizontal cylinders (I and II), each cylinder can be either extended or retracted. Four possible combinations allow us to position the hand over each of the four tracks.

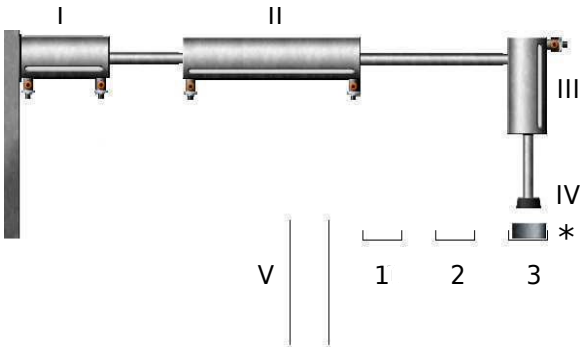


Fig. 2. Pick-and-Place manipulator system implementation with three cylinders

The hand also has a vertical cylinder (III) which is used for moving a suction unit (IV) up and down. The suction unit is used to grab work pieces (*) from the input tracks. The PnP manipulator is also equipped with sensors which detect whether its cylinders are in extended or retracted positions and sensors which detect whether there is a work piece on each of the input tracks. There is also a sensor which detects whether something has been grabbed with the suction unit. The manipulator is used for moving work pieces from input tracks to the output track.

A. Plant and environment surrogate model

Here we describe how the surrogate plant-environment model was created. For each input track we create two Boolean variables p_i and pp_i. We give no restriction of the values of p_i. Its value is arbitrary at any time and models the environment actions (in this case, the user). The value of pp_i is determined in the following way. If the suction unit is currently grabbing the work piece from the i-th track, pp_i is set to $false$, otherwise if $pp_i = true$, we preserve its value. In all other cases $pp_i := p_i$. This approach allows us to model situations when work pieces can appear on tracks at arbitrary time, but ones appeared stay there until being grabbed with the suction unit.

To model the suction unit we need just one Boolean variable vs, indicating whether it is on or off. If the controller says to turn it on or off, we do it, otherwise we leave its value unchanged.

Since cylinders are moving from retracted position to extended and back not instantly, for each cylinder we create a variable $cs_i : \{cst_j | j = 0 \dots 3\}$, indicating its current state. We update these variables according to the commands the controller produces. The sensors indicate whether the cylinder is in retracted or extended position depending on the value of cs_i.

The last thing we have to do is to set up a sensor indicating whether the suction unit is grabbing something or not. To do this we create a Boolean variable vac. The algorithm to recalculate its value is simple. If the suction unit is off then $vac := false$, if the suction unit is placed over a track

with a work piece, the vertical cylinder is extended and the suction unit is on then $vac := true$, otherwise its value is left unchanged.

The whole plant-environment model is then working in the following way. On each step we produce an input event, requesting the manipulator to recalculate its state. This is done according to the ECC, current model state and current values of input variables. Then we recalculate the state of the environment according to its current state, output variables produced by the ECC, and rules above.

B. Experiments

Our experiments pursued several goals. First, to check and demonstrate the feasibility of synthesizing function block logic from scenarios and temporal properties. Second, to see if using counterexample length in the fitness function is a good idea or not (which is not obvious).

For simplicity we used one test scenario that corresponds to picking up a single work piece from the first input track. This scenario has a total length of 2339. The following three LTL properties were considered:

1) $G(\neg(c1Extend \wedge c1Retract))$ – safety property that states that cylinder I must never be given commands to extend and retract simultaneously;
2) $G(\neg(c2Extend \wedge c2Retract))$ – analogous safety property about cylinder II;
3) $G(pp1 \rightarrow F(vp1))$ – functional property stating that if a work piece appears on the first input track, it will eventually be picked up by the manipulator.

The algorithm was implemented in Java. We used NuSMV 2.5.4 for temporal properties verification and ran it with default parameters. All experiments we executed on a machine with a 64-core AMD Opteron™6378 @ 2.4 GHz processor, the parallel search algorithm was run using 16 cores and a maximum of 32 Gb of RAM.

First, we synthesized ECCs using only the test scenario as input and performed 50 independent runs. Then we ran our method with LTL properties verification, but without using the counterexample length in the fitness function. Finally, we ran our method using the counterexample length fitness function component. Results of experimental runs are summarized in Table I.

First, we shall note that all synthesized ECCs satisfy the first two LTL properties. This was to be expected, since these two properties are *safety* properties – they describe what the system *should not* do. They can be satisfied even if the ECC does not do anything at all.

Next, none of the 50 ECCs synthesized without the use of verification in the process of synthesis satisfied the third, functional property. This confirms that it is highly unlikely to accidently generate an ECC that complies with a functional temporal property.

On the other hand, all runs of configurations 2 and 3 which used our verification-aware fitness, resulted in ECCs that satisfied all three given temporal formulas. It is also evident that adding temporal formulas to input data increases the mean running time needed to find a solution up to an order of magnitude. Next, using the counterexample length (configuration 3) decreases the average running time by about 30%.

After synthesis all ECCs were tested in the FBDK[3] environment. All ECCs synthesized using verification-aware fitness functions allowed to pick up the first work piece several times (tested five times for each ECC), while all the ECCs generated only from scenarios failed to process the work piece more than one time.

V. Discussion & Conclusion

In this work we presented an approach for basic IEC 61499 function block reconstruction which ensures that the synthesized FB satisfies the same LTL properties as the original FB. We incorporate FB verification into fitness function evaluation and a mutation operator. Verification is done in a closed-loop manner using a manually constructed surrogate model of the environment and the plant. This model only reflects the external behavior of the plant and thus is much simpler than the exact model.

We were able to synthesize controllers for the PnP manipulator that reflect one of its behaviors – picking up a single work piece from the first input track and moving it to the output track. By adding verification of several LTL properties to the synthesis process, we ensured that all ECCs are able to process the first work piece multiple times.

The surrogate model closed-loop verification is simultaneously the biggest advantage and disadvantage of the proposed approach: for plants more complex than the considered PnP manipulator it may be hard to create. However, once created, it gives a huge increase in verification speed. It may be possible to create such surrogate models automatically, this is a topic for future work.

Another way is to work with full plant models and derive a heuristic procedure that would allow to increase verification speed on early stages of the search. For instance, we can use bounded model checking [15] and increase the value of the bound as solutions get better.

Another moment not considered in the experimental section is parameter tuning. Algorithm parameters greatly influence its performance. We believe that in the future *online* parameter tuning can be used to adapt parameter values of our algorithm during different phases of the search. This is another direction of future research.

Future work also includes evaluating the approach on larger/harder specifications.

Acknowledgements

This work was financially supported by the Government of Russian Federation, Grant 074-U01, and also partially funded by RFBR according to the research project No. 16-37-00205 mol_a.

[3]http://www.holobloc.com/doc/fbdk

TABLE I
EXPERIMENTAL RESULTS

#	Configuration	Time, s				Satisfied all LTL
		min	mean	median	max	
1	Scenario	32	85	81	280	0%
2	Scenario + LTL (no F^{ce}_{smv})	222	752	656	1689	100%
3	Scenario + LTL	164	563	561	1575	100%

REFERENCES

[1] V. Vyatkin, "IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 768–781, Nov 2011.

[2] D. Chivilkhin, A. Shalyto, S. Patil, and V. Vyatkin, "Reconstruction of function block logic using metaheuristic algorithm: Initial explorations," in *Proceedings of IEEE International Conference on Industrial Informatics*, 2015, pp. 1239–1242.

[3] T. Back, D. B. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*. Bristol, UK: IOP Publishing Ltd., 1997.

[4] R. Hametner, B. Kormann, B. Vogel-Heuser, D. Winkler, and A. Zoitl, "Test case generation approach for industrial automation systems," in *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*, 2011, pp. 57–62.

[5] V. Vyatkin and H.-M. Hanisch, "Verification of distributed control systems in intelligent manufacturing," *Journal of Intelligent Manufacturing*, vol. 14, no. 1, pp. 123–136, 2003.

[6] V. Dubinin, V. Vyatkin, and H.-M. Hanisch, "Modelling and verification of iec 61499 applications using prolog," in *IEEE Conference on Emerging Technologies and Factory Automation*, 2006, pp. 774–781.

[7] S. Patil, D. Drozdov, V. Dubinin, and V. Vyatkin, *Technological Innovation for Cloud-Based Engineering Systems: 6th IFIP WG 5.5/SOCOLNET Doctoral Conference on Computing, Electrical and Industrial Systems, DoCEIS 2015, Costa de Caparica, Portugal, April 13-15, 2015, Proceedings*. Cham: Springer International Publishing, 2015, ch. Cloud-Based Framework for Practical Model-Checking of Industrial Automation Applications, pp. 73–81. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-16766-4_8

[8] V. Vyatkin, H.-M. Hanisch, C. Pang, and C.-H. Yang, "Closed-loop modeling in future automation system engineering and validation," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 39, no. 1, pp. 17–28, Jan 2009.

[9] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, 2003.

[10] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

[11] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking," in *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, ser. LNCS, vol. 2404. Copenhagen, Denmark: Springer, 2002.

[12] S. Patil, V. Vyatkin, and M. Sorouri, "Formal verification of Intelligent Mechatronic Systems with decentralized control logic," in *Proceedings of the 17th IEEE Conference on Emerging Technologies Factory Automation*, 2012, pp. 1–7.

[13] F. Tsarev and K. Egorov, "Finite state machine induction using genetic algorithm based on testing and model checking," in *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, ser. GECCO '11. New York, NY, USA: ACM, 2011, pp. 759–762.

[14] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1987, pp. 14–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=42512.42515

[15] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," *ADV COMPUT, Elsevier*, vol. 58, pp. 117–148, 2003.