Inferring Automata-Based Programs from Specification With Mutation-Based Ant Colony Optimization

Daniil Chivilikhin
ITMO University
Computer Technologies Laboratory
Saint Petersburg, Russia
chivdan@rain.ifmo.ru

Vladimir Ulyantsev
ITMO University
Computer Technologies Laboratory
Saint Petersburg, Russia
ulyantsev@rain.ifmo.ru

ABSTRACT

In this paper we address the problem of constructing correct-by-design programs with the use of the automata-based programming paradigm. A recent algorithm for learning finite-state machines (FSMs) MuACO*sm* is applied to the problem of inferring extended finite-state machine (EFSM) models from behavior examples (test scenarios) and temporal properties, and is shown to outperform the genetic algorithm (GA) used earlier.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming

Keywords

Design/Synthesis, Software Engineering, Empirical study

1. INTRODUCTION

Designing a complex software system is hard especially in such areas as aviation, space industry, and banking, where high software reliability is required. Verification must be used to check such software, since testing can only reveal existing errors but cannot guarantee their absence. One verification method is model checking [3], where temporal properties are checked for a model of software. This brings simplicity but creates a gap between software and its model.

Automata-based programming [6] allows to overcome this limitation. Programs designed using this paradigm can be automatically transformed to Kripke models used in model checking [7]. Automata-based programming proposes to implement software systems as sets of interacting *automated-controlled objects*. An automated-controlled object consists of an extended finite-state machine (EFSM) controller and a controlled object, which is characterized by a set of commands implemented by its methods. It can have a number of associated *event suppliers* which provide input for it. Upon receiving an event from some event supplier, the EFSM makes a transition, sending a sequence of output actions (commands) to the controlled object, which processes these commands and calls its corresponding methods.

A key advantage of automata-based programming is that it allows to infer correct-by-design control programs from behavior examples using search-based optimization (e.g. evolutionary algorithms [1]) and model checking. The main issue here is to design an efficient optimization algorithm well fit for EFSM inference. In this work we apply the recent FSM induction algorithm MuACO*sm* [2], which has been shown to outperform GA on simple problems, to the more practical problem of learning EFSMs from specification consisting of both test scenarios and temporal properties.

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
ACM 978-1-4503-2881-4/14/07.
http://dx.doi.org/10.1145/2598394.2598446.

2. PROBLEM STATEMENT

In this paper we define an EFSM as a septuple $\langle S, s_0, Z, \Sigma, \Delta, \delta, \lambda \rangle$, where S is a set of *states*, $s_0 \in S$ is the *initial state*, Z is a set of Boolean *input variables*, Σ is a set of *input events*, Δ is a set of *output actions*, $\delta\colon S \times \Sigma \times 2^Z \to S$ is the *transitions function* and $\lambda\colon S \times \Sigma \times 2^Z \to \Delta^*$ is the *actions function*.

Various examples of program behavior may be used, however here we consider test scenarios only. A *test scenario* γ_i is a sequence of *scenario elements* $\langle e, \varphi, O \rangle$, where $e \in \Sigma$, $\varphi \in 2^Z$ is a Boolean formula over the input variables, and $O \in \Delta^*$. An EFSM complies with a scenario element $\langle e, \varphi, O \rangle$ in state s if it contains a transition labeled with event e, sequence of output actions O, and a *guard condition* equal to φ as a Boolean formula. An EFSM complies with a test scenario if it complies with all scenario elements in the corresponding states when the scenario is processed sequentially.

Following [7] we use LTL for temporal properties representation. The LTL language consists of propositional variables *Prop*, logical operators "and", "or", "not", and a set of temporal operators, such as **G** (**G**lobally in the future), **X** (ne**X**t), **F** (in the **F**uture), **U** (**U**ntil) and **R** (**R**elease). Here, propositional variables are wasEvent($\cdot$) and wasAction($\cdot$) terms for each event and output action, correspondingly.

The problem of specification-based EFSM inference is to find an EFSM with a maximum of N_{states} states compliant with a set of test scenarios $\Gamma = \{\gamma_i\}$ and a set of LTL formulae.

3. EFSM INFERENCE METHODS

In this work we apply the recent MuACO*sm* algorithm [2] to the stated problem and compare it with the previously used GA [7]. EFSM *skeletons* [7] are used instead of EFSMs

for solution representation, *smart transition labeling* [7] is applied prior to each fitness evaluation to transform a skeleton into an EFSM.

We use a fitness function proposed in [7]. Its first component f_{sc} evaluates compliance of the EFSM with scenarios. Each scenario is processed separately. Scenario elements are passed to the EFSM and the resulting output sequence $\hat{A}_i$ is recorded and compared with the reference output sequence A_i. The overall expression for f_{sc} has the form:

$$f_{\text{sc}} = \frac{1}{|\Gamma|} \sum_{i=1}^{|\Gamma|} \left(1 - \frac{\text{ED}\left(\hat{A}_i, A_i\right)}{\max\left(\text{len}(\hat{A}_i), \text{len}\left(A_i\right)\right)} \right),$$

where $|\Gamma|$ is the number of test scenarios, $\text{len}(s)$ is the length of sequence s and $\text{ED}\left(s_1, s_2\right)$ is the edit distance [4] between sequences s_1 and s_2.

The second component f_{LTL} evaluates the compliance of the EFSM with LTL properties. The verifier developed by the authors of [7] allows to mark EFSM transitions that are certainly not included in the counterexample – *checked* transitions. The expression for f_{LTL} is:

$$f_{\text{LTL}} = \frac{1}{m} \sum_{i=1}^{m} \frac{t^i_{\text{checked}}}{t_{\text{reachable}}},$$

where m is the number of LTL formulae, t^i_{checked} is the number of checked transitions for the i-th formula, and $t_{\text{reachable}}$ is the total number of transitions reachable from the initial state.

The final expression for the fitness function takes into account the number of transitions in the EFSM and has the form:

$$f = f_{\text{sc}} + f_{\text{LTL}} + \frac{M - n_{\text{tran}}}{100 \cdot M},$$

where n_{tran} is the total number of EFSM transitions and M is a number sufficiently larger than n_{tran}.

4. EXPERIMENTS

To compare algorithms fairly we first used the `irace` [5] package to tune parameters of GA and MuACO*sm*. A set of 100 random problem instances has been generated for tuning. The process was allotted 12 hours for each algorithm, the resulting parameter values were used in all following experiments.

Experiments were conducted on inferring EFSM controllers for an elevator doors model [7] (9 scenarios, 11 LTL formulae, FSMs with 6 states) and an alarm clock model (38 scenarios, 9 LTL formulae, FSMs with 4 states)[1].

For each algorithm two experiments were performed on inferring EFSMs from scenarios only and also from scenarios together with LTL formulae. Each experiment was repeated 1000 times, each run continued until reaching an optimal solution ($f = 2.0075$ for the elevator doors and $f = 2.0086$ for the alarm clock). Mean and standard errors of used fitness evaluation numbers are presented in Table 1, best values for each case are highlighted in bold. These results indicate that MuACO*sm* finds the optimal solution 5–15 times faster than GA. The statistical significance of these results was validated using the Wilcoxon statistical test [8],

[1]All source code, scripts and other data used in the experiments is included in supplementary material and is also available at https://code.google.com/p/muaco/

Table 1: Mean numbers of fitness evaluations used until reaching optimal solution

	Elevator	**Alarm clock**
Scenarios, GA	877 (330)	45714 (44160)
Scenarios, MuACO	**131 (38)**	**3011 (2719)**
Scenarios+LTL, GA	63199 (65321)	95516 (99036)
Scenarios+LTL, MuACO	**10997 (10260)**	**5797 (5075)**

which gave p-values less than 2.2×10^{-6} for both "Scenarios" and "Scenarios+LTL" cases.

5. CONCLUSION

We have applied the MuACO*sm* algorithm to the problem of inferring EFSMs from specification. It was found that for the examined elevator doors and alarm clock control problems MuACO*sm* yields a significantly better performance than that of GA.

Acknowledgements

This work was financially supported by the Government of Russian Federation, Grant 074-U01, and also partially supported by RFBR, research project No. 14-07-31337 мол_а.

6. REFERENCES

[1] T. Back, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK, UK, 1st edition, 1997.

[2] D. Chivilikhin and V. Ulyantsev. MuACOsm: A New Mutation-Based Ant Colony Optimization Algorithm for Learning Finite-State Machines. In *Proceedings of the fifteenth annual conference on Genetic and evolutionary computation*, GECCO '13, pages 511–518, New York, NY, USA, 2013. ACM.

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

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

[5] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, Iterated Race for Automatic Algorithm Configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

[6] A. Shalyto and N. Tukkel'. SWITCH Technology: An Automated Approach to Developing Software for Reactive Systems. *Programming and Computer Software*, 27(5):260–276, 2001.

[7] 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*, GECCO '11, pages 759–762, New York, NY, USA, 2011. ACM.

[8] F. Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, Dec. 1945.