ИЗВЕСТИЯ ВЫСШИХ УЧЕБНЫХ ЗАВЕДЕНИЙ
Т. 55, № 8/3 ФИЗИКА 2012

УДК 519.7

R.F. GALIMULLIN

ON SEPARATING UML SPECIFIED NON-DETERMINISTIC FINITE STATE MACHINES WITH TIME-OUTS

UML state machine diagrams are briefly discussed and two approaches for deriving separating sequences for timed Finite State Machines are compared.

Keywords: *non-deterministic finite state machines with time-outs, state machine diagram.*

Introduction

Nowadays the complexity of various discrete systems, such as computers, cell phones, embedded controllers, etc., is extremely high, and thus, it is essential to provide high-quality testing at every stage of the system development. Moreover, since technical failures are very expensive, the system testing and verification at early stages, especially at the design stage, is of great importance [1]. Many of such systems are formally described using the UML (the Unified Modeling Language) [2]. The UML being a visual modeling language allows obtaining the comprehensive and detailed information about a system under design, as well as provides a possibility for convenient update of the system. Correspondingly, the UML is widely used in software engineering, business project development, hardware design and in a number of other applications. It is also fruitful to use the UML for the integration of fresh employees and cross-national collaboration. There are some approaches for verifying systems described by UML. One of them is based on extracting a formal model from the UML description. In our paper, we use a timed finite-state machine model (TFSM) that is an extension of a classical finite-state machine (FSM). By definition, a FSM is described as a finite set of states and transitions between them. Every transition is labeled by an 'input/output' pair, where an input triggers the transition and an output is a system response to a given input. A FSM is deterministic if at each state for each input, there is at most one transition labeled by this input, otherwise, a FSM is non-deterministic. If transitions depend on a time of applying an input, FSM is called timed FSM, and in our paper we deal with a certain type of timing constraints, so-called time-outs [3]. If no input is applied at a current state during an appropriate time period (time-out), the timed FSM can move to another prescribed state. In this paper, we propose an approach how to distinguish two timed FSMs extracted from UML descriptions.

UML Statecharts

In this section we briefly describe UML notations. Being a multipurpose language the UML allows using state machine diagrams that are perfectly tailored to the FSM representation. Each state machine diagram is supposed to include at least one state that can be additionally specified by the entry, current or exit activity (*entry*/*do*/*exit*). A transition may be either blank or can be labeled with a guard condition, a timeout, or both of them. The UML description may contain some special states, so-called pseudostates, that are used for initiating or finishing the execution (*initiation pseudostate*, *final state*, *terminate node*), for joining or forking transitions (*fork vertex*, *join vertex*), etc. In order to extend state machine possibilities composite states are added. Composite states contain one (*orthogonal composite state*) or more regions which are executed simultaneously. An output of a composite state may include a join vertex. In this case the transition is executed when one of the regions reaches a final state. Otherwise, *wait all* condition becomes true. Composite states are very useful due to the ability to model and test a system at different levels of abstraction, i.e., it is possible to model certain parts of a system as well as the overall system. Nevertheless, it is unknown how to distinguish two UML diagrams, e.g., the specification and a certain implementation. However, as it was mentioned above, state machine diagrams allow the transformation of UML descriptions to timed, possibly non-deterministic, FSMs. In order to distinguish two timed FSMs separating sequences are constructed [4].

Deriving Separating Sequences for FSMs with timeouts

Two approaches for deriving separating sequences were elaborated. The first approach is based on a proper modification of the algorithm for deriving a separating sequence for classic FSMs. It is based

on a truncated successor tree constructed for the intersection of two timed FSMs describing the common behavior of both machines. For a current node, specified by the set of pairs $<state_i, time_i>$ there are edges labeled by each input, and for each input the set of all possible successors labels a node at the next level. A node is terminal if there are no successors for the given input (Rule 1), or the set of corresponding items contains a set of items of the node at a higher level of the tree (Rule 2). If the tree has a node that is terminal due to Rule 1, a separating sequence exists. A separating sequence is a sequence of inputs labeling a path from the root to the terminal node tailed by the current input. For TFSMs the algorithm is modified by adding extra edges labeled by the minimum of time-outs. Thus, a separating sequence for two timed FSMs consists of <*input*, *time*> pairs. The second approach does not need any modification of the above mentioned algorithm. Instead of this, there is an additional step, namely, the transformation of a timed FSM into a classical FSM. The main point is that time-labeled transition (*n* time instances, for instance) is represented as *n* sequential transitions with a special input 1 (wait for one time instance) and a special output *N*. Correspondingly, the number of states increases compared with a timed FSM. After the transformation the intersection of two FSMs is constructed in order to derive a separating sequence. Comparing two approaches following remarks can be made. The successor tree for the first approach seems to be smaller, as a corresponding FSM has more states than the initial TFSM. On the other hand, the software implementation of the second approach is rather simpler and thus, we may expect that a separating sequence can be derived faster.

For more detailed comparison, both approaches were implemented and computer experiments with software implementations were performed. TFSMs with 20–100 states and 10–750 inputs were randomly generated. For each pair of parameters 10 pairs of TFSMs were generated and average length of a separating sequence (if sequence exists) was calculated. The results are following (Fig. 1 and 2).

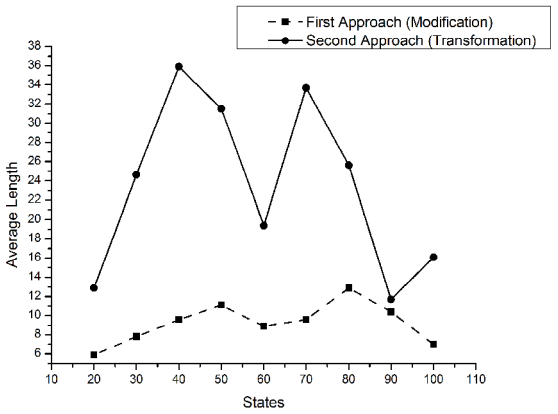

Fig. 1. States – length plot.

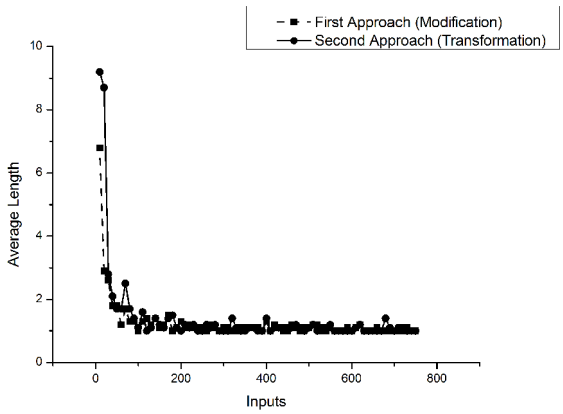

Fig. 2. Inputs – length plot.

The experimental results show, that for randomly generated TFSMs separating sequences derived using the first approach are shorter. Moreover, when the number of input increases almost every randomly generated TFSM has a separating sequence of length 1.

REFERENCES

1. Karpov Y. Model Checking. – BHV-Saint-Petersburg, 2010. – 552 p. (in Russian)
2. Booch G. The Unified Modeling Language User Guide, Second Edition / Jacobson I., Rumbaugh J. – Addison Wesley, 2005. – 496 p.
3. Gromov M., El-Fakih K., Shabaldina N., and Evtushenko N. // Formal Techniques for Distributed Systems. – Germany: Springer, 2009. – P. 137–151.
4. Galimullin R. and Shabaldina N. // Proceedings of the 5th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2011). – Ekaterinburg, USU, 2011. – P. 100–104.

Tomsk State University, Tomsk, Russia
E-mail: nihilkhaos@gmail.com

Поступила в редакцию 15.06.12.

Rustam Galimullin, student.