AsmetaA: Animator for Abstract State Machines

Silvia Bonfanti[1](✉), Angelo Gargantini[1], and Atif Mashkoor[2,3]

[1] Department of Economics and Technology Management, Information Technology and Production, University of Bergamo, Bergamo, Italy
{silvia.bonfanti,angelo.gargantini}@unibg.it

[2] Software Competence Center Hagenberg GmbH, Hagenberg, Austria
atif.mashkoor@scch.at

[3] Johannes Kepler University Linz, Linz, Austria
atif.mashkoor@jku.at

Abstract. In this paper, we present AsmetaA – a graphical animator for Abstract State Machines integrated within the ASMETA framework. The execution of formal specifications through animation provides several advantages, e.g., it provides an immediate feedback about system behavior, it helps understand system evolution, and it increases the overall acceptability of formal methods.

1 Introduction

One important feature of the Abstract State Machines (ASM) method [3] is that it allows to *execute* specifications that represent the evolution of a system by a *sequence of states*. An important advantage of the state-based execution is that it helps users understand through experimentation the behavior of the system being designed [7].

The ASMETA framework [4] provides an environment for systems development using ASMs including a simulator. During simulation, the user can interact with the simulator by inserting monitored values when required and observe the system evolution. The user can drive and follow the ASM execution and understand whether the specification really captures the intended system behavior. However, the simulation engine currently available for the ASMETA platform provides only a textual interface that prints the states as strings with some optional messages on the console. Observing system evolution in this fashion can be difficult. For this reason, the need for a graphical animator for the ASMETA platform has been felt for a long time.

The graphical animation of specifications consists in showing by means of graphical elements, e.g., tables and colors, the evolution of the system state.

The research reported in this paper has been partly supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH.

M. Butler et al. (Eds.): ABZ 2018, LNCS 10817, pp. 369–373, 2018.
https://doi.org/10.1007/978-3-319-91271-4_25

This provides several advantages: the user can perform a rapid validation, it helps in understanding the system behavior, and it shows concrete scenarios in which abstract states can be instantiated. For this reason, many formal notations and tools (see Sect. 4) support this kind of validation technique.

In this paper, we present AsmetaA – a graphical animator for the ASMETA platform, which shows the system execution and state evolution using graphical elements. It is integrated in the framework and it can be downloaded and installed as an eclipse plug-in[1]. The rest of the paper is organized as follows: In Sect. 2, we discuss the main goal of this work. In Sect. 3, we briefly present the AsmetaA tool. A brief comparison of AsmetaA with similar tools is presented in Sect. 4. The paper is concluded in Sect. 5.

2 Animation of ASMs: Requirements and Goals

In the wake of a recent effort for providing visual information to users of the ASMETA framework, we have already developed a visualizer that provides a graphical view of ASMETA models [2]. The visualizer provides information about the structure of the machine, in terms of a set of construction rules and schemas that give a graphical representation of an ASM and its rules. However, in current settings, information about system dynamics is missing. For this reason, we started to work on the concept of *animation* of ASM specifications. In our tool, *animation* has the following main objectives:

1. Providing a user with complete information about all the locations in one *state*. In this way, the user can understand the system state at every step.
2. Showing the *evolution* of an ASM during the execution. In this way, the user can understand the behavior of the specification.
3. Using colors, tables, and figures over simple text to convey information about states and their evolution.

In order to achieve these goals, we decided to structure the animator as shown in Fig. 1. The table captures the two dimensions of locations in one state and their evolution. On the horizontal axis, we want to represent the evolution of the execution, by showing the sequence of states. On the vertical axis, we want to show the states, i.e., locations and their values at each state.

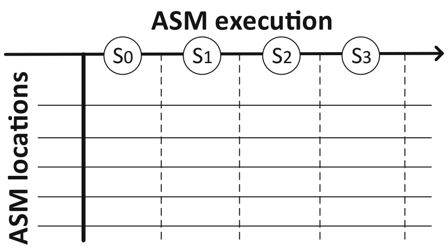
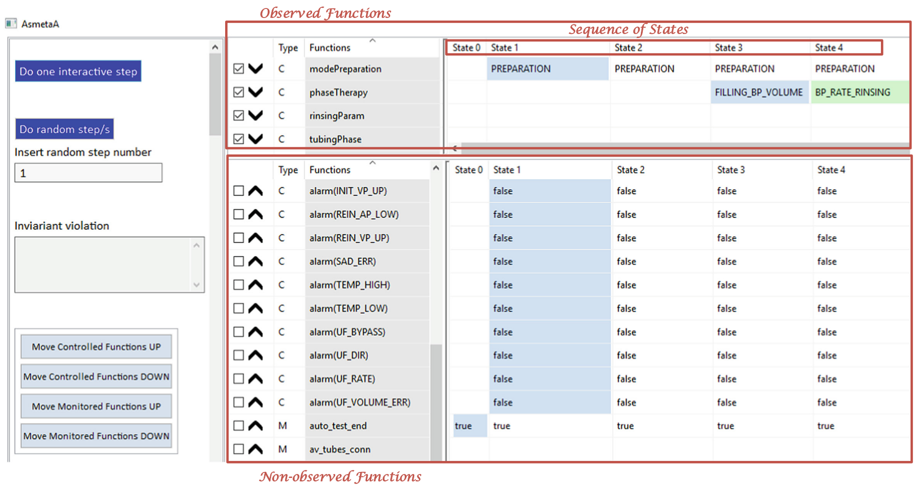


Fig. 1. Animation of an ASM

As an auxiliary goal of the animator, we use graphical elements also for user interaction and avoid the use of textual consoles whenever possible. Additionally, we cope with the specification complexity by allowing the user to highlight some locations of interest.

[1] http://asmeta.sourceforge.net/.


AsmetaA
Observed Functions
Sequence of States
Do one interactive step
Do random step/s
Insert random step number
1
Inviariant violation
Move Controlled Functions UP
Move Controlled Functions DOWN
Move Monitored Functions UP
Move Monitored Functions DOWN
Non-observed Functions

	Type	Functions	State 0	State 1	State 2	State 3	State 4
☑	C	modePreparation		PREPARATION	PREPARATION	PREPARATION	PREPARATION
☑	C	phaseTherapy				FILLING_BP_VOLUME	BP_RATE_RINSING
☑	C	rinsingParam					
☑	C	tubingPhase					

	Type	Functions	State 0	State 1	State 2	State 3	State 4
☐	C	alarm(INIT_VP_UP)		false	false	false	false
☐	C	alarm(REIN_AP_LOW)		false	false	false	false
☐	C	alarm(REIN_VP_UP)		false	false	false	false
☐	C	alarm(SAD_ERR)		false	false	false	false
☐	C	alarm(TEMP_HIGH)		false	false	false	false
☐	C	alarm(TEMP_LOW)		false	false	false	false
☐	C	alarm(UF_BYPASS)		false	false	false	false
☐	C	alarm(UF_DIR)		false	false	false	false
☐	C	alarm(UF_RATE)		false	false	false	false
☐	C	alarm(UF_VOLUME_ERR)		false	false	false	false
☐	M	auto_test_end	true	true	true	true	true
☐	M	av_tubes_conn					

Fig. 2. AsmetaA tool

3 AsmetaA: Animator for ASM Specifications

A graphical view of the AsmetaA tool is shown in Fig. 2. We now describe various characteristics of this tool.

Random vs interactive animation. Random and interactive animations are two modalities of execution provided by the animator. Random animation runs the ASM specification and the values of monitored functions are chosen by the animator. The number of steps is selected by the user and it can be changed dynamically. Interactive animation runs one step at a time and the value of monitored functions are selected by the user. These two modalities of animation can be mixed within the same run. This means that one state can be reached using random animation and the next state can be reached using interactive animation. The two modalities of animation are executed using the following corresponding buttons.

– *Do one interactive step*: asks the user about the value of monitored functions and runs one step. The monitored functions are inserted through a dialog box.
– *Do random step/s*: runs one or more steps based on the number inserted in the field *Insert random step number*.

In case of invariant violation, the message is shown in the dedicated text box.

Random simulation: multiple steps. With complex specifications, running one random step each time is tedious. To overcome this limit, we have added a field in which the user inserts the number of steps to be performed and the tool performs the random simulation accordingly.

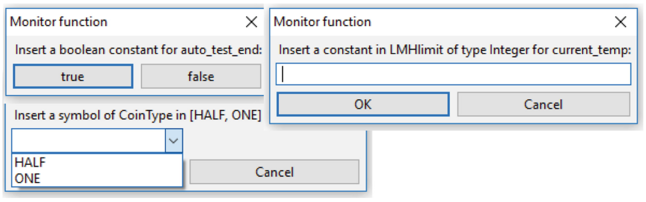

Monitor function
Insert a boolean constant for auto_test_end:
true
false
Insert a symbol of CoinType in [HALF, ONE]
HALF
ONE
Cancel
Monitor function
Insert a constant in LMHlimit of type Integer for current_temp:
OK
Cancel

Fig. 3. GUI dialogs allow the user to input new values for monitored functions

Use of tables. The first version of the animator was realized using one table for all functions. However, it was difficult for the user to follow the functions of his/her interest in complex specifications. To cope with this difficulty, we have now added two tables. The upper table contains the functions observed by the user, while the lower table contains all other functions. When the function appears for the first time during the simulation, the animator inserts it in the lower table. If the user is interested to follow this function, he/she has to move it to the upper table using the check box display in the first column. When the user is no longer interested in observing a specific function, he/she can move it into the lower table and still follow other observed functions. The user can also move functions (from one table to the other) that belong to a specific type (controlled functions or monitored functions) using the buttons in the lower left corner shown in Fig. 2. Moreover, the content of the tables can be sorted alphabetically based on the type or name of the functions.

Colors for easy reading. One of the main features of AsmetaA is the usage of multiple colors to facilitate the readability of tables. Multiple colors help a user to identify particular events during the ASM execution: the initial value of the function (light blue cells) and when the function changes the value compared to the previous state (light green cells).

Dialog box. The insertion of monitored functions is achieved through different dialog boxes (see Fig. 3) depending on the type of function to be inserted. For example, in case of a boolean function, the box has two buttons: one if the answer is true and one if the answer is false. By pushing the button, the user assigns the corresponding value to the function. In case of functions with the enumerative domain, the dialog box shows all the possible assignable values and the user selects the chosen value from a combo box. At the moment, for other data types, the user inserts the value in a text box.

4 Related Work

Several formal methods support the animation of specifications. For the B family methods, one of the main tools that provides such facility is ProB [6]. In this tool, the user can select the events to fire while the state is constantly updated and shown to the user. Differently from our animator, ProB displays the history of events but the history of the system states does not appear.

Visualization of traces is used in TLA+ to show counter examples in case of errors while model checking a specification [8]. Also for the NuSMV model checker, traces can be shown in tables by the NuSeen toolbox [1]. As compared to model checkers, our animator is intended to be used in an interactive way to allow the validation of specifications.

There is already one tool with the goal of animating ASM specifications [5]. In this work, the authors extend CoreASM with some plug-ins to show the state evolution of the ASM specifying a flash file system. As compared to AsmetaA, it is an application specific work focusing on a particular case study. AsmetaA, on the other hand, is a generic animator capable of animating *any* ASM. As a future work, we plan to introduce a method allowing the definition of special graphical widgets for application specific animations.

5 Conclusion

In this paper, we have presented AsmetaA – an animation engine for ASM specifications. The tool supports users in the validation process and concretizes the abstract states. The users can run two types of animation: random and interactive. In initial tests, the graphical interface of AsmetaA has been proved intuitive, simple, and user friendly.

References

1. Arcaini, P., Gargantini, A., Riccobene, E.: NuSeen: a tool framework for the NuSMV model checker. In: 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST), pp. 476–483 (2017)
2. Arcaini, P., Bonfanti, S., Gargantini, A., Riccobene, E.: Visual notation and patterns for abstract state machines. In: Milazzo, P., Varró, D., Wimmer, M. (eds.) STAF 2016. LNCS, vol. 9946, pp. 163–178. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50230-4_12
3. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-642-18216-7
4. Gargantini, A., Riccobene, E., Scandurra, P.: A metamodel-based language and a simulation engine for abstract state machines. J. UCS **14**(12), 1949–1983 (2008)
5. Haneberg, D., Junker, M., Schellhorn, G., Reif, W., Ernst, G.: Simulating a flash file system with CoreASM and eclipse. In: Informatik 2011: Informatik schafft Communities, Beiträge der 41. Jahrestagung der Gesellschaft für Informatik e.V. (GI). LNI, vol. 192, p. 355. GI (2011)
6. Leuschel, M., Butler, M.: ProB: an automated analysis toolset for the B method. J. Softw. Tools Technol. Transf. **10**(2), 185–203 (2008)
7. Mashkoor, A., Jacquot, J.-P.: Validation of formal specifications through transformation and animation. Requir. Eng. **22**(4), 433–451 (2017)
8. Yu, Y., Manolios, P., Lamport, L.: Model checking TLA+ specifications. In: Pierre, L., Kropf, T. (eds.) Correct Hardware Design and Verification Methods, pp. 54–66. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48153-2_6