Statechart Analysis with Symbolic PathFinder

Corina S. Păsăreanu
CMU-SV/NASA Ames
NASA Ames Research Park, MS 269-2, PO Box 1
Moffett Field, CA 94035
Email: corina.s.pasareanu@nasa.gov

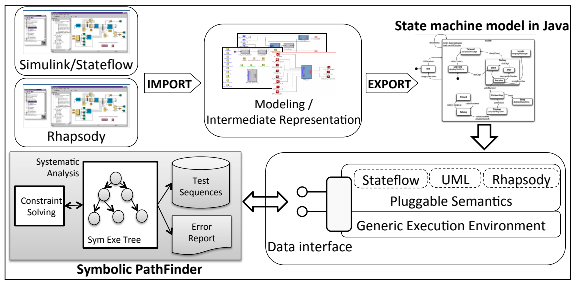

Figure 1. The Polyglot Framework

I. Statechart Modeling with Polyglot

We report here on our on-going work that addresses the automated analysis and test case generation for software systems modeled using multiple Statechart formalisms. The work is motivated by large programs such as NASA Exploration, that involve multiple systems that interact via safety-critical protocols and are designed with different Statechart variants. To verify these safety-critical systems, we have developed Polyglot [1], a framework for modeling and analysis of model-based software written using different Statechart formalisms. Polyglot uses a common intermediate representation with customizable Statechart semantics and leverages the analysis and test generation capabilities of the Symbolic PathFinder tool [2]. Polyglot is used as follows (see Figure 1). First, the structure of the Statechart model (expressed in Matlab Stateflow or Rational Rhapsody) is translated into a common intermediate representation (IR). The IR is then translated into Java code that represents the structure of the model. The semantics are provided as "pluggable" modules.

Currently, Polyglot includes modules that implement the semantics of Matlab Stateflow, Rational Rhapsody, and UML Statemachines; the framework can be extended easily with other statechart semantics. The Java code representing the structure of the model is combined with one of these semantic modules, resulting in an executable component. Analysis and test case generation is then performed using Symbolic PathFinder (SPF). Polyglot can be used to execute and analyze systems that contain interacting components modeled with the different Statechart formalisms.

II. Analysis with Symbolic PathFinder

Symbolic PathFinder (SPF) [2] is an analysis tool for Java bytecode that performs symbolic execution to generate test cases that achieve high test coverage. Symbolic execution is a systematic program analysis that uses symbolic values instead of actual data inputs and symbolic expressions to represent the values of program variables. The state of a symbolically executed program includes the symbolic values of program variables, a path condition (PC), and a program counter. The path condition is a Boolean formula over the symbolic inputs, encoding the *constraints* which the inputs must satisfy in order for an execution to follow the particular associated path. These conditions are solved using off-the-shelf constraint solvers to generate test cases guaranteed to exercise the analyzed code. Symbolic PathFinder generates both test vectors and test sequences; the latter are necessary for testing looping, reactive programs, such as the ones translated from the Statechart models. During test case generation, SPF checks the properties of the code, expressed as assertions or property automata.

We have applied SPF to the analysis and test case generation of Statecharts in Polyglot. The analysis uncovered subtle interaction errors between components modeled with Statecharts, for the flight software developed for NASA Exploration. To increase the speed of our analysis, we are investigating program specialization via symbolic execution. This involves using SPF to specialize the Polyglot semantic modules with respect to particular Statechart models. Our preliminary results are encouraging, showing 3x improvement in analysis time, with 10x fewer instructions being executed by SPF.

Acknowledgment

This is joint work with D. Balasubramanian, M. Whalen, G. Karsai, T. Pressburger and M. Lowry.

References

[1] D. Balasubramanian, C. S. Păsăreanu, M. W. Whalen, G. Karsai, and M. R. Lowry, "Polyglot: modeling and analysis for multiple statechart formalisms," in *ISSTA*, 2011, pp. 45–55.

[2] C. S. Păsăreanu, P. C. Mehlitz, D. H. Bushnell, K. Gundy-Burlet, M. Lowry, S. Person, and M. Pape, "Combining unit-level symbolic execution and system-level concrete execution for testing nasa software," in *ISSTA*, 2008, pp. 15–26.