A Step Towards Verification and Synthesis from Simulink/Stateflow Models*

Karthik Manamcheri
manamch1@illinois.edu

Sayan Mitra
mitras@illinois.edu

Stanley Bak
sbak2@illinois.edu

Marco Caccamo
mcaccamo@illinois.edu

ABSTRACT

This paper describes a toolkit for synthesizing hybrid supervisory control systems starting from the popular Simulink/Stateflow modeling environment. The toolkit provides a systematic strategy for translating Simulink/Stateflow models to hybrid automata and a discrete abstraction-based algorithm for synthesizing supervisory controllers.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Model Checking*

General Terms

Algorithms, Verification

Keywords

simulink, stateflow, hybrid automata

1. INTRODUCTION

A key barrier towards applying hybrid system design and analysis techniques to engineering problems is the steep learning curve of the existing hybrid tools. To address this issue, we are building a toolkit that connects Mathwork's popular Simulink/Stateflow (SLSF) environment with new and existing hybrid system tools. While our overarching goals are similar to those of several other projects (see, for example, [9, 8, 6]), the technical approaches are distinct. For example, the synthesis algorithms in [6] rely on approximate bisimulations while ours is based on traditional simulation. Checkmate [8] provides custom modeling blocks within SLSF, whereas our approach allows the use of commonly used Simulink blocks. Deferring a systematic comparison of these approaches for a later paper, here we introduce two core components of our toolkit: (a) HyLink: a tool that transforms a restricted class of SLSF models to hybrid automata for verification, and (b) SimplexGen: a tool that generates supervisory controller logic for a class of SLSF models. These tools can be downloaded from [1].

Figure 1(a) shows the workflow of our tool suite. The SLSF environment provides the front-end for developing and simulating the models which are read by HyLink to produce an internal representation called Hybrid Intermediate Representation (HIR). Different code generation modules analyze the HIR and generate specifications for analysis and synthesis tools, such as HyTech [5], UPPAAL [4], and SimplexGen [2].

*This project is partially funded by John Deere Technology Innovation Center, Champaign, IL and National Science Foundation (NSF) under the grant CNS-1016791.

HSCC'11, April 12–14, 2011, Chicago, Illinois, USA.

2. HYLINK: SLSF TO HYBRID AUTOMATA

A Simulink model describes a dynamical system as a collection of interconnected functional blocks. Stateflow is used to model hierarchical state machines with discrete transitions and continuous flows, and they can be interconnected with other Simulink blocks.

An Example. We consider the model of an autonomous waypoint tracking system (WTS). A *planner* provides the vehicle *controller* a sequence of way points on the two-dimensional plane. The controller periodically sets the acceleration and steering of the vehicle based on the current position, heading, and the waypoint. The vehicle moves according to certain differential equations with the above inputs. The system is said to be *safe* if the vehicle remains within a specified bound of the line joining the current and the previous waypoints (see Figure in 1(c)). Figure 1(b) shows the Simulink model of the vehicle, which is a part of the SLSF model for the complete system. The output of the sub-system in Figure 1(b), reflects the functions which calculate the coordinates x and y of the vehicle.

We have identified a restricted class of Stateflow models which can be translated to hybrid automata. These models requires, for example, that (a) the transition guards to be closed sets, (b) the reset map for an incoming transition at a discrete state (location) and the invariant for that location to overlap. Stateflow models do not have explicit invariants associated with locations. However, as all transitions are urgent, an implicit invariant is derived as follows: the closure of the complement of each outgoing transition guard is computed and the intersection of these sets gives the invariant. Establishing the formal relationship between the set of executions produced by the original Stateflow model and that of our translated hybrid automaton is ongoing work.

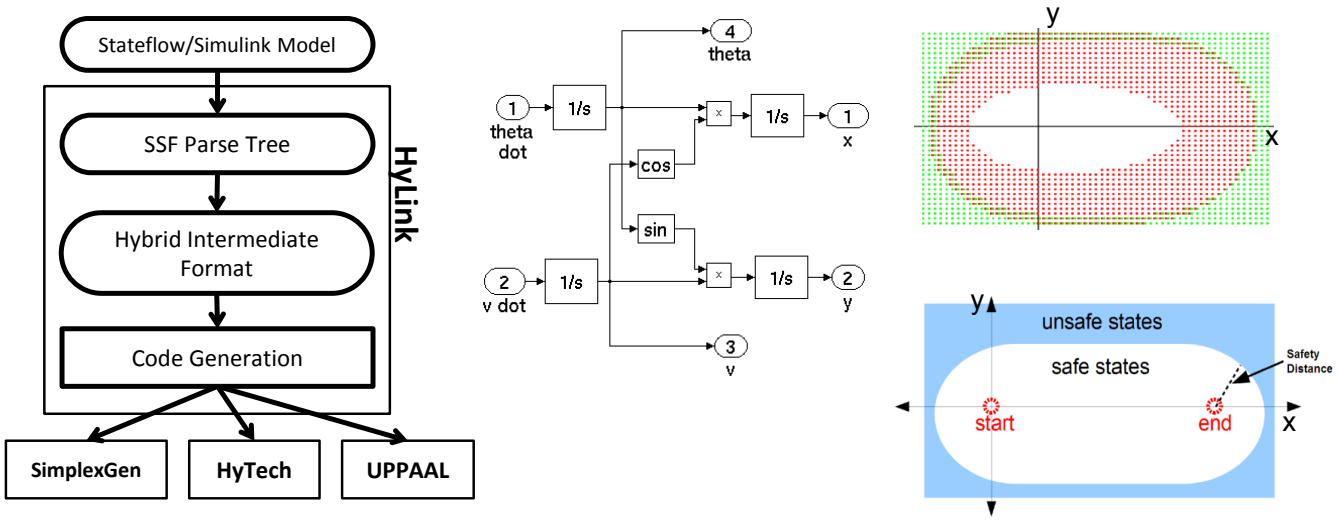

Figure 1: (a) Toolkit work flow, (b) Simulink model of the WTS, (c) Top: Unsafe regions (green points) and switching set (red points) generated by SimplexGen for WTS, Bottom: Single-waypoint system with unsafe and safe regions

For Simulink models that do not have any Stateflow components, HyLink translation to hybrid automaton proceeds as follows: For a model with n switch blocks, a hybrid automaton with 2^n locations is produced—one for each possible configuration of the switches. For each location, the differential and algebraic equations guiding the evolution of the (continuous) variables are derived by composing the functions of the individual blocks that are relevant in the corresponding switch configuration. The invariant condition for the locations are derived by composing the functions of the blocks which drive the switches. The generated hybrid automata have switched modes with invariants but transition conditions and resets cannot be extracted from the Simulink model.

SLSF models do not permit nondeterminism which is a serious restriction in modeling concurrent systems. We address this by developing an idiom: introduction of redundant input variables. For the purposes of simulation these variables are connected to *nondeterminism resolver blocks (NRB)* which set the inputs either according to some deterministic function or using random numbers. During tranalsation, these inputs are ignored to produce a nondeterminisitic hybrid automata.

2.1 Acknowledgments

We thank Daniel Grier for developing the translator from HIR to HyTech, which revealed to us several subtleties in Stateflow semantics.

3. GENERATING SWITCHING CONTROLLER

The Simplex Architecture [7] provides a way to safely sandbox untrusted controllers [3]. The *complex controller* (CC), is combined with a safety wrapper which monitors the state of the plant and can switch to a simpler *safety controller* (SC) before the system state becomes unrecoverable. The switching boundary where the Simplex system should switch from complex to safety controller is given by [2, 3]:

$$G_c = \delta\text{-}BackReach_{\mathcal{HC}}(BackReach_{\mathcal{HS}}(\mathcal{U})),$$

where $\mathcal{U}$ is unsafe set, BackReach$_{\mathcal{HS}}$ is the backwards reachable set with the SC and δ-BackReach$_{\mathcal{HC}}$ is the bounded backwards reachable set with the CC within δ time, and δ is the period of the logic.

The SimplexGen tool overapproximates these $BackReach$ sets by constructing a discrete abstraction of the hybrid automaton and using bounds on the derivatives for each variable also extracted from the automaton model. The maximum and minimum derivatives are then be used in the discrete abstraction to overapproximate the reach set of the original automata. SimplexGen takes as input the hybrid automata for the CC and SC, an equation describing the unsafe set, and outputs the safety wrapper, namely the switching logic of the supervisory controller. In Figure 1(c)(top), the computed switching set (red) and the unsafe set (green) are shown for a fixed velocity and heading for the WTS example.

4. REFERENCES

[1] Hylink tool. http://hsver.crhc.illinois.edu.
[2] S. Bak, A. Greer, and S. Mitra. Hybrid cyberphysical system verification with simplex using discrete abstractions. In *RTAS '10*, 2010.
[3] S. Bak, K. Manamcheri, S. Mitra, and M. Caccamo. Sandboxing controllers for cyber-physical systems. In *ICCPS'11*, 2011.
[4] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL in 1995. In *TACAS*, pages 431–434, 1996.
[5] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *CAV '97*.
[6] M. M. Jr., A. Davitian, and P. Tabuada. Pessoa: A tool for embedded control software synthesis. In *CAV*, 2010.
[7] L. Sha. Using simplicity to control complexity. *IEEE Software*, 18:20–28, 2001.
[8] B. I. Silva, K. Richeson, B. H. Krogh, and A. Chutinan. Modeling and verification of hybrid dynamical system using checkmate. In *ADPM*, 2000.
[9] A. Tiwari. Formal semantics and analysis methods for Simulink Stateflow models. Technical report, SRI International, 2002.