Symbolic Analysis for Improving Simulation Coverage of Simulink/Stateflow Models

Rajeev Alur
University of Pennsylvania
alur@cis.upenn.edu

Aditya Kanade
University of Pennsylvania
kanade@seas.upenn.edu

S. Ramesh
GM India Science Lab
ramesh.s@gm.com

K.C. Shashidhar
GM India Science Lab
shashidhar.kc@gm.com

ABSTRACT

Aimed at verifying safety properties and improving simulation coverage for hybrid systems models of embedded control software, we propose a technique that combines numerical simulation and symbolic methods for computing state-sets. We consider systems with linear dynamics described in the commercial modeling tool Simulink/Stateflow. Given an initial state x, and a discrete-time simulation trajectory, our method computes a set of initial states that are guaranteed to be equivalent to x, where two initial states are considered to be equivalent if the resulting simulation trajectories contain the same discrete components at each step of the simulation. We illustrate the benefits of our method on two case studies. One case study is a benchmark proposed in the literature for hybrid systems verification and another is a Simulink demo model from Mathworks.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools*; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms

Design, Experimentation, Measurement, Verification

Keywords

Simulink, Stateflow, Hybrid systems, Simulations, Coverage

1. INTRODUCTION

Model-based design offers a promising approach for detecting and correcting errors in early stages of system design (cf. [30, 25]). In this methodology, a designer first constructs a model, with mathematically precise semantics, of the system under design, and performs extensive analysis with respect to correctness requirements before generating the implementation from the model. The appropriate mathematical model for embedded control systems is *hybrid systems* that combines the traditional state-machine based models for discrete control with classical differential- and algebraic-equations based models for continuously evolving physical activities [4]. Such models can capture both the controller — the system under design, and the plant — the environment in which the system operates. The hybrid systems model can be subjected to two kinds of analyses: simulation and verification.

In simulation, a possible execution of the model upto a finite time horizon is obtained using numerical methods. Simulation-based analysis is a well-accepted industrial practice, and is typically applicable to complex models. The drawback of the method is that it cannot handle nondeterminism well. Sources of nondeterminism in hybrid systems include inputs, initial states, and noise in the plant dynamics. Running multiple simulations cannot guarantee absence of errors, and unlike testing of hardware and software, there is little work on quantifying the coverage obtained by multiple simulations of dynamical or hybrid systems.

In verification, the goal is to check all possible executions of the system using symbolic model checking or deductive proof methods (see [22, 18, 9, 24, 26, 13] for sample approaches). A central focus of this line of research is on algorithms for verifying safety properties by computing a symbolic representation of the set of reachable states. There is a lot of ongoing research on representing state-sets by alternatives such as polyhedra, ellipsoids, and zonotopes, and on abstraction methods such as predicate abstraction, bisimulation-preserving dimensionality reduction, counter-example guided abstraction refinement, and qualitative differential equations [5, 10, 35, 15]. Despite these efforts, scalability of the model checking tools for hybrid systems remains limited. This has prompted some researchers to explore if simulation can be augmented with symbolic techniques to improve its effectiveness [23, 16, 11, 27], and our approach belongs to this category. It is worth noting that in software analysis, a recent trend is to combine testing with symbolic model checking, and has resulted in very powerful tools for debugging large programs [17, 33].

Our analysis technique is currently aimed at hybrid systems for which the dynamics in each discrete mode of operation is described as a linear system, and the only source of nondeterminism is in the choice of the initial state. In this setting, once we fix the parameters for the numerical sim-

EMSOFT'08, October 19–24, 2008, Atlanta, Georgia, USA.

ulation, namely, the time horizon, the simulation step-size, and the choice of the integration routine, the execution $\rho(x)$ is completely determined by the initial state x. We call two initial states x and y equivalent if the executions $\rho(x)$ and $\rho(y)$ agree on the discrete component of the state at each step. For verifying safety properties (under the discrete-time simulation semantics) and for simulation coverage, it would be redundant to run simulations from equivalent initial states. Our core analysis algorithm, given an input state x and the execution $\rho(x)$, computes a set X of initial states that is guaranteed to contain only states equivalent to x, and thus, can be ignored for choosing initial states in subsequent simulations. This computation involves backwards preimage computation possibly using underapproximations, and our current implementation, based on Parma Polyhedra Library [6], uses convex polyhedra as a representation for state-sets.

The proposed equivalence relation can be used in two ways for improving the effectiveness of simulations: first, it provides an effective way of coverage of the initial states and second, it helps in optimizing the number of simulations by keeping only one representative test case for each equivalence class of initial states.

Another goal of this paper is to develop techniques that can analyze models described in the commonly used commercial tool Simulink/Stateflow. Most of the academic tools analyze models described in some specialized notation for hybrid systems. This poses a challenge to apply these tools to industrial examples. Towards this goal, we describe a reasonably detailed mapping, which can potentially be automated, from Simulink/Stateflow models to hybrid systems. For our experiments, we use the commercial simulator from Mathworks to obtain executions, and apply the symbolic analysis to manually translated models.

We report experimental results on two case studies: the *room heating benchmark* from the hybrid systems verification benchmarks [12], and the *vehicle climate control* whose description has a variety of features of Simulink/Stateflow language [1]. We give some representative coverage results. Our results demonstrate that a small number of simulations can lead to a significant coverage of the initial state-space. For example, in the room heating benchmark, 5 simulations lead to estimated 83% coverage, and 20 simulations lead to estimated 90% coverage; the corresponding numbers for the second case study are 18% and 31%. We also analyze the effect of length of simulation on coverage and run-time of our tool. For example, increasing the length of individual simulations from 10 to 50 steps for the room heating benchmark gave estimated 35% and 43% coverage for 5 and 20 simulations respectively.

Related work.

Analysis techniques for computing coverage of initial state-space using bisimulation metrics or expansion functions are used in test generation [23] and verification [16, 11]. These techniques identify constraints under which continuous trajectories starting from two states stay sufficiently close together. Since we focus on the discrete-time simulation semantics, our problem is computationally less demanding, and can be fully automated. We also note that these earlier approaches currently do not allow transition guards to be convex polyhedra, and as a result, cannot handle the Simulink/Stateflow case studies considered in this paper.

There exist many commercial and in-house test generation tools for Simulink designs which include Reactis [28], STB [34], T-VEC [38], Simulink Design Verifier [32], Beacon Tester [8], and AutoMOTgen [14]. Given a model and a set of coverage goals over model elements (like blocks, branches, and states), these tools use combinations of randomization and constraint solving techniques to generate tests. The kind of coverage over the initial state-space that we attempt is finer than the coverage acheived by these tools. The test cases generated by our tool, though more in number, cover the model much more extensively and can potentially reveal more bugs. In particular, our method rules out only those trajectories that are close (in the sense of sequences of discrete states visited) to the already observed ones and thus guaranteed to detect any violation of temporal logic requirements, upto the specified bound on simulation steps.

In these proceedings, [31] presents a testing method for covering the Stateflow elements. A combination of random, directed, backtracking, and feedback-based testing and a set of heuristics are used to achieve better coverage of the Stateflow transitions that involve non-linear constraints. This method is similar to the above discussed methods (and hence differs from our approach for the same reason) but reports better coverage than these methods on many models.

The modeling language of Simulink/Stateflow tool lacks a formal and rigorous definition of its semantics. Several types of formal semantics are presented for Stateflow, namely, denotational [19], operational [20], and communicating pushdown automata based [36]. Specialized tools are developed for translating subsets of the language to hybrid automata [3], Lustre [37], SMV [7], and SAL [14]. We give a systematic translation from Simulink/Stateflow models to hybrid systems and also formalize the discrete-time simulation semantics of Simulink for the hybrid system models.

Organization.

In Section 2, we define a class of hybrid systems, called linear hybrid systems. We present the discrete-time simulation semantics and define the notion of equivalence of states. In Section 3, we give an algorithm to compute equivalent states. We explain translation from Simulink models to hybrid systems in Section 4. We describe the case studies and the experimental results in Sections 5 and 6.

2. FORMAL MODELING

In this section, we define a class of hybrid systems called, linear hybrid systems, which have linear dynamics for each discrete state. We define the discrete-time simulation semantics of hybrid systems and the notion of equivalence of states in terms of resulting simulation trajectories.

2.1 Linear hybrid systems

We first introduce some terminology. Consider a set of real-valued variables $X = \{x_1, \ldots, x_n\}$. An n-dimensional linear expression $e\colon \mathbb{R}^n \to \mathbb{R}$ over X is of the form $\sum_{i=1}^{n} a_i x_i + a_{n+1}$ where $a_i \in \mathbb{R}$ for each $i \in \{1, \ldots, n+1\}$. Let E_n denote the set of all n-dimensional linear expressions over X.

An n-dimensional linear predicate $u\colon \mathbb{R}^n \to \mathbb{B}$ over X is of the form $e \bowtie 0$ for a linear expression e over X and a relation $\bowtie \in \{\geq, >\}$. Let P_n be the set of finite sets of n-dimensional linear predicates over X, where an element of P_n denotes the conjunction of its elements and represents an n-dimensional convex polyhedron.

DEFINITION 1. *A* linear hybrid system (LHS) *is specified as a tuple* $H = (Q, X, Init, Flow, Jump)$ *as follows:*

- *A finite set* Q *of discrete states (modes).*
- *A finite set* $X = \{x_1, \ldots, x_n\}$ *of real-valued variables.*
- *A function Init which associates an initial continuous state-space* $Init(q) \in P_n$ *with each discrete state* $q \in Q$.
- *A function Flow which associates a linear expression* $Flow(q, x_i) \in E_n$ *with each discrete state* $q \in Q$ *and each variable* $x_i \in X$ *defining the derivative of* x_i *at* q.
- *A finite set Jump of discrete transitions of the form* (q, q', g, r) *where* $q \in Q$ *is the source state,* $q' \in Q$ *is the target state,* $g \in P_n$ *is a guard, and* $r(x_i) \in E_n$ *is a reset map which associates a linear expression with each variable* $x_i \in X$. *We assume that the guards on outgoing transitions from* q *are pairwise disjoint.*

Consider an LHS $H = (Q, X, Init, Flow, Jump)$. The *state-space* of H is $\mathcal{S} = Q \times \mathbb{R}^n$. A *state* $s \in \mathcal{S}$ of H is a pair $(q, \tilde{\mathrm{v}})$ where $q \in Q$ is a discrete state and $\tilde{\mathrm{v}} = \langle v_1, \ldots, v_n \rangle \in \mathbb{R}^n$ is a *continuous state* denoting a valuation of all the variables $x_i \in X$. The *initial state-space* $\mathcal{S}_0 \subseteq \mathcal{S}$ of H is the set of all pairs $(q, \tilde{\mathrm{v}})$ such that the continuous state $\tilde{\mathrm{v}} \in Init(q)$.

The *discrete transition relation* of H is a set $\mathcal{J} \subseteq \mathcal{S} \times \mathcal{S}$ that defines transitions between states of H. If the system is in a state $(q, \tilde{\mathrm{v}})$ and $\tilde{\mathrm{v}} \in g$ ($\tilde{\mathrm{v}}$ satisfies the guard g) for some transition $(q, q', g, r) \in Jump$ then the system instantaneously jumps to the discrete state q' at which the continuous state $\tilde{\mathrm{v}}$ is reset to $\tilde{\mathrm{v}}'$ such that $v'_i = r(x_i)[x \mapsto \tilde{\mathrm{v}}]$ where $x \mapsto \tilde{\mathrm{v}}$ is the substitution of v_i for x_i for all indices i.

In hybrid systems models, invariant sets are associated with discrete states to allow non-determinism in the time at which a discrete transition occurs. In Simulink/Stateflow models there is no non-determinism in system evolution and discrete transitions are urgent. The case when the system stays in the same discrete state is modeled by self-loops in the transition relation *Jump*.

2.2 Discrete-time simulation semantics

Simulink updates the continuous state of a model by numerically integrating its derivative at discrete-time steps. Since our objective is to analyze Simulink/Stateflow models by formalizing them as (linear) hybrid systems, we define discrete-time simulation semantics of hybrid systems based on the simulation semantics of Simulink.

In Simulink, the numerical integration is performed by an integration routine $\mathbb{S}$. In this paper, we consider (explicit) fixed-step integration routines (solvers). An example is the *Euler* routine: $Euler(x_i, e_i, h) := x_i + h\, e_i$ where $x_i \in X$, the derivative of x_i is $e_i \in E_n$, and $h \in \mathbb{R}_+$ is the step size.

Given an integration routine $\mathbb{S}$ and a step size h, the discrete-time evolution of the continuous state of an LHS H at a discrete state q is defined as: $x'_i := Evol_q^{\mathbb{S},h}(x_i) := \mathbb{S}(x_i, Flow(q, x_i), h)$. We now define the discrete-time simulation semantics of a hybrid system as follows.

DEFINITION 2. *Consider an LHS* H, *an integration routine* $\mathbb{S}$, *a time step* h, *and a simulation length* $k \in \mathbb{N}$. *The set* $Traj_k^{\mathbb{S},h}$ *of* k*-step* simulation trajectories *of* H *consists of sequences of the form* $\langle s_0, s'_0, \ldots, s_i, s'_i, \ldots, s_{k-1}, s'_{k-1} \rangle$ *where the states* $s_i = (q_i, \tilde{\mathrm{v}}_i)$ *and* $s'_i = (q'_i, \tilde{\mathrm{v}}'_i)$ *are defined as follows:*

- *The state* $(q_0, \tilde{\mathrm{v}}_0) \in \mathcal{S}_0$ *is an initial state.*
- *For* $i \in \{0, \ldots, k-1\}$, $(q'_i, \tilde{\mathrm{v}}'_i)$ *is the state after the continuous state evolution of the system from* $(q_i, \tilde{\mathrm{v}}_i)$ *in one simulation step i.e.* $q'_i = q_i$ *and* $\tilde{\mathrm{v}}'_i = \langle v'_1, \ldots, v'_n \rangle$ *where* $v'_j = Evol_{q_i}^{\mathbb{S},h}(x_j)[x \mapsto \tilde{\mathrm{v}}_i]$.

  For $i \in \{0, \ldots, k-2\}$, $((q_i, \tilde{\mathrm{v}}'_i), (q_{i+1}, \tilde{\mathrm{v}}_{i+1})) \in \mathcal{J}$.

An ith state $s_i = (q_i, \tilde{\mathrm{v}}_i)$ denotes the state of the hybrid system at a time instance $t_i = i\,h$. The evolution of the continuous state and the discrete state transitions are deterministic. Therefore, for any initial state, the simulation trajectory is uniquely defined.

We assume that the step size h is small enough so that multiple discrete transitions do not take place in one simulation step and no discrete transition is missed while simulating. In Simulink, taking the simulation step to be the greatest integer divisor of the sampling times of all the discrete blocks, ensures this (ref. Section 4.3).

2.3 Improving coverage of simulations

Simulation is a preferred analysis method in industry for understanding and validation of system designs. However, the simulation methodology infamously suffers from the lack of completeness with respect to the coverage of state-space or system behaviors. Further, the effectiveness of simulations crucially depends on the choice of initial states. The selection of initial states is usually semi-automatic (user-guided) or random, demanding significant user involvement or resulting in redundant simulations.

We propose the following steps to improve effectiveness of simulations: (1) To identify initial states that would lead to redundant simulations and (2) To select simulation inputs so that complete coverage of simulation trajectories or equivalently the entire initial state-space is achieved, potentially leading to design verification. Towards this, we first define the notion of equivalence of states.

DEFINITION 3. *Consider an LHS* H, *an integration routine* $\mathbb{S}$, *a time step* h, *and a simulation length* k. *Consider the simulation trajectories* $\langle s_0, s'_0, \ldots, s_{k-1}, s'_{k-1} \rangle$ *and* $\langle z_0, z'_0, \ldots, z_{k-1}, z'_{k-1} \rangle$ *of the system starting from the states* s_0 *and* z_0 *respectively. Let* $s_i = (q_i, \tilde{\mathrm{v}}_i)$ *and* $z_i = (p_i, \tilde{\mathrm{w}}_i)$. *The states* s_0 *and* z_0 *are* equivalent, *i.e.* $Equiv_k^{\mathbb{S},h}(s_0, z_0) = true$, *iff the discrete states of the respective system trajectories are equal i.e.* $q_i = p_i$ *for each* $i \in \{0, \ldots, k-1\}$.

Simulation coverage can be improved by using the following scheme. Initially, simulate a given system for k steps starting from some initial state and record the simulation trajectory. Using the simulation trajectory, determine initial states that are equivalent (upto k-steps) to the chosen initial state. The input for the next simulation is then selected from outside the already covered initial states, potentially leading to a distinct simulation trajectory. The process can be repeated until the entire initial state-space is covered or violation of some safety property is observed.

The proposed technique requires an algorithm for computation of equivalent states with respect to a simulation trajectory. More formally, given an LHS H and a k-step simulation trajectory $\langle s_0, s'_0, \ldots, s_{k-1}, s'_{k-1} \rangle$, we need to determine the set of initial states $A_0 \subseteq \mathcal{S}_0$ such that for any $z_0 \in A_0$, $Equiv_k^{\mathbb{S},h}(s_0, z_0) = true$.

Algorithm 1: Computation of equivalent states

Input : An LHS H, a routine $\mathbb{S}$, a time step h, and a trajectory $\langle (q_0, \tilde{\mathrm{v}}_0), \ldots, (q'_{k-1}, \tilde{\mathrm{v}}'_{k-1})\rangle$

Output: A set $A_0 \subseteq \mathcal{S}_0$ of states equiv. to $(q_0, \tilde{\mathrm{v}}_0)$

1 $B := \mathbb{R}^n$
2 $B := PreE(q_{k-1}, B)$
3 **for** $i := k-1;\ i > 0;\ i := i-1$ **do**
4 | **let** $t = (q_{i-1}, q_i, g, r) \in Jump$ **s.t.** $\tilde{\mathrm{v}}'_{i-1} \in g$
5 | $B := PreR(t, B) \cap g$
6 | $B := PreE(q_{i-1}, B)$
7 $A_0 := (\{q_0\} \times B) \cap \mathcal{S}_0$
8 **return** A_0

3. ALGORITHM FOR COMPUTATION OF EQUIVALENT STATES

Consider an LHS $H = (Q, X, Init, Flow, Jump)$, an integration routine $\mathbb{S}$, a time step h, and a k-step trajectory given by $\langle (q_0, \tilde{\mathrm{v}}_0), (q'_0, \tilde{\mathrm{v}}'_0), \ldots, (q_{k-1}, \tilde{\mathrm{v}}_{k-1}), (q'_{k-1}, \tilde{\mathrm{v}}'_{k-1})\rangle$. To compute a set A_0 of initial states of H that are equivalent to $(q_0, \tilde{\mathrm{v}}_0)$, we compute a set A_i of states that are equivalent to $(q_i, \tilde{\mathrm{v}}_i)$ with respect to the suffix $\langle (q_i, \tilde{\mathrm{v}}_i), \ldots, (q'_{k-1}, \tilde{\mathrm{v}}'_{k-1})\rangle$ of the given trajectory, for $i = k-1$ to 0.

We first define two functions, *PreE* and *PreR* as follows. Given a set of continuous states $B \subseteq \mathbb{R}^n$ and a discrete state q, $PreE(q, B)$ computes an underapproximation of the preimage of B under the continuous state evolution of the discrete state q. Formally, if $\tilde{\mathrm{v}} \in PreE(q, B)$ then $\tilde{\mathrm{v}}' = \langle v'_1, \ldots, v'_{k-1}\rangle \in B$ where $v'_j = Evol_q^{\mathbb{S},h}(x_j)[x \mapsto \tilde{\mathrm{v}}]$. Given a discrete transition $t = (q, q', g, r) \in Jump$, $PreR(t, B)$ computes an underapproximation of the preimage of B under the reset map r. Formally, if $\tilde{\mathrm{v}} \in PreR(t, B)$ then $\tilde{\mathrm{v}}' = \langle v'_1, \ldots, v'_{k-1}\rangle \in B$ where $v'_j = r(x_j)[x \mapsto \tilde{\mathrm{v}}]$.

From Definition 3, we know that for all states $(p, \tilde{\mathrm{w}}) \in A_i$, the discrete state $p = q_i$. Consider the set of continuous states $B_i \subseteq \mathbb{R}^n$ such that $A_i = \{(q_i, \tilde{\mathrm{w}})\colon \tilde{\mathrm{w}} \in B_i\}$. Let $t = (q_{i-1}, q_i, g, r) \in Jump$ such that $\tilde{\mathrm{v}}'_{i-1} \in g$. The transition t exists and is unique as the guards of the outgoing transitions from q_{i-1} are pairwise disjoint.

During simulation, the continuous state $\tilde{\mathrm{v}}_{i-1}$ has been updated to $\tilde{\mathrm{v}}'_{i-1}$ according to the continuous state evolution of q_{i-1}, $\tilde{\mathrm{v}}'_{i-1}$ was found to satisfy the guard of a transition, and was reset to $\tilde{\mathrm{v}}_i$ according to the reset map of the transition. Thus, using the functions *PreR* and *PreE* described above, we can compute a set B_{i-1} of continuous states such that $A_{i-1} = \{(q_{i-1}, \tilde{\mathrm{w}}_{i-1})\colon \tilde{\mathrm{w}}_{i-1} \in B_{i-1}\}$. Algorithm 1 performs the backward propagation of sets of equivalent states.

Let i be the counter for the loop (lines 3–6). In any iteration i, depending on the continuous state $\tilde{\mathrm{v}}'_{i-1}$, Algorithm 1 identifies the transition between the discrete states q_{i-1} and q_i that was taken by the hybrid system during the ith step of the simulation. Suppose the transition is $t = (q_{i-1}, q_i, g, r)$ and the set of equivalent states for the suffix $\langle (q_i, v_i), \ldots, (q'_{k-1}, v'_{k-1})\rangle$ of the input trajectory computed by the algorithm is B_i. In Step 5, a preimage of B_i with respect to the reset map r (using *PreR*) followed by an intersection with the guard g is computed. In Step 6, the preimage of the set with respect to the continuous state evolution $Evol_{q_{i-1}}^{\mathbb{S},h}$ is computed (using *PreE*). Let B_{i-1} be the resulting set. It is easy to verify that for any state in $\{(q_{i-1}, \tilde{\mathrm{w}}_{i-1})\colon \tilde{\mathrm{w}}_{i-1} \in B_{i-1}\}$, the system simulated according to Definition 2, leads to a state in $\{(q_i, \tilde{\mathrm{w}}_i)\colon \tilde{\mathrm{w}}_i \in B_i\}$ in one time step. This gives us soundness of the algorithm.

THEOREM 4. *For an LHS H, an integration routine $\mathbb{S}$, a time step h, and a trajectory $\langle s_0, \ldots, s'_{k-1}\rangle$, for the set A_0 computed by Algorithm 1, $\forall z_0 \in A_0, Equiv_k^{\mathbb{S},h}(s_0, z_0) = true$.*

Our current implementation of Algorithm 1 uses convex polyhedra for representing state-sets and performs the required operations using Parma Polyhedra Library. We discuss the implementation details in Section 5.3.

4. TRANSLATION OF SIMULINK MODELS TO LINEAR HYBRID SYSTEMS

The modeling language of Simulink/Stateflow tool lacks a formal and rigorous definition of its semantics. In this section, we present our estimate of the semantics and accordingly present a translation scheme from Simulink/Stateflow models to linear hybrid systems. We use a simplified version of the vehicle climate control (VCC) model (one of the case studies) as an example. The model is shown in Figure 1.

4.1 Simulink/Stateflow models

A Simulink/Stateflow design is represented graphically as a diagram consisting of inter-connected Simulink blocks. It represents the time-dependent mathematical relationships between the inputs, states, and outputs of the design.

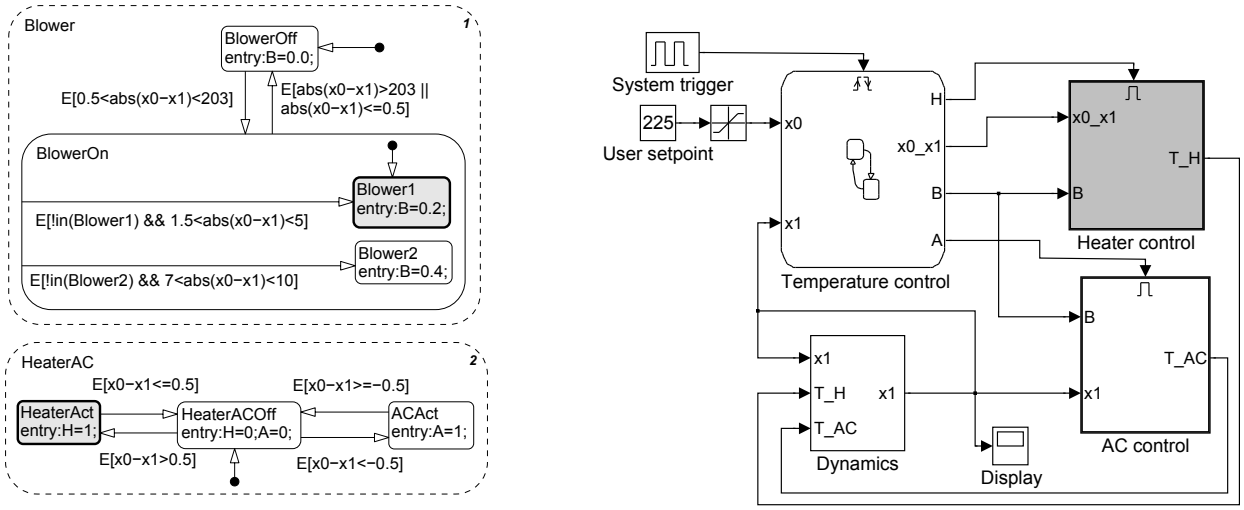
DEFINITION 5. *A* Simulink model $SL = (D, B, C, L)$ *consists of the following components:*

- *A finite set D of typed* variables *partitioned into input variables D_I, state variables D_S, auxiliary variables D_A, and output variables D_O.*
- *A finite set B of Simulink blocks. Each block has input, output, and local variables. The input and output variables are associated with input and output ports. A non-input variable can be defined by a block as a function of the non-output variables. A Simulink block can itself be a Simulink diagram.*
- *An ordered relation $C \subseteq B \times B$ that represents* connections *between the blocks. A connection $c = (b, b') \in C$ connects an output port of b to an input port of b' and represents the flow of data between the corresponding variables of b and b'.*
- *A function $L\colon C \to D_A$ associates a unique auxiliary variable to each connection.*

Simulink provides a diverse family of continuous, discrete, and logical building blocks. Stateflow complements these features by providing another class of Simulink blocks, called Stateflow charts, for modeling mode control logic. Stateflow charts are hierarchical state machines which are specified in a variant of Statecharts [21].

DEFINITION 6. *A* Stateflow chart $SF = (V, E, S, TR, T)$ *consists of the following components:*

- *A finite set V of typed* variables *partitioned into input variables V_I, output variables V_O, and local variables V_L.*



(a) Stateflow chart: Temperature control

(b) Simulink model

Figure 1: Simplified and partial model of vehicle climate control (VCC) system

- *A finite set E of* events *partitioned into input events E_I, output events E_O, and local events E_L.*
- *A finite set S of* states *partitioned into atomic, AND, and (exclusive) OR states. If an AND state is active then all its component states become active. If an OR state is active then exactly one of its component states becomes active.*

  Each state $s \in S$ is labeled with a finite (possibly empty) set of actions A_s. An action is an assignment to a non-input variable or an event broadcast. The set of actions A_s is partitioned into ordered sets of actions upon entry to s: entry(s)*, actions prior to exit from s:* exit(s)*, and actions while the state s is active:* during(s)*.*
- *A relation $TR \subseteq S \times S$ represents the hierarchical composition of the states. The graph (S, TR) is a* tree*. A tree edge $(s, s') \in TR$ denotes that s' is a component state of s. The atomic states in S are the only leaf nodes of the tree.*

  An active configuration *of the states of the chart can be represented as a tree (S_{Act}, TR_{Act}) where $S_{Act} \subseteq S$ and $TR_{Act} \subseteq TR$. The tree edges in TR_{Act} respect the relation of the composite (non-leaf) states and their substates depending on the type (AND or OR) of the composite states.*
- *A finite set T of* transitions*. A transition $t \in T$ is a tuple $t = (s, s', e, \varphi, act_{condition}, act_{transition})$ where $s \in S$ is the source state, $s' \in S$ is the target state, $e \in E$ is an enabling event, φ is a guard which is a well-formed quantifier-free formula in predicate logic over the variables V, $act_{condition}$ are actions executed as soon as φ is evaluated as true, and $act_{transition}$ are actions executed before the state s' is entered.*

Consider the Simulink model shown in Figure 1(b). The user setpoint x_0 is the input variable of the model and the internal temperature x_1 is the state variable i.e. $D_I = \{x_0\}$ and $D_S = \{x_1\}$. The model consists of Temperature control, Heater control, AC control, and Dynamics blocks which are inter-connected as shown. Heater control and AC control blocks are enabled subsystem blocks and are respectively enabled by the signals H and A from the Stateflow chart.

Temperature control chart shown in Figure 1(a) is a Stateflow chart and implements a supervisory control. The input variables to it are $V_I = \{x_0, x_1\}$ and the output variables are $V_O = \{H, A, B, x_0\_x_1\}$. The variable B determines the blower output and the variable $x_0\_x_1 = x_0 - x_1$.

Temperature control chart is an AND state consisting of Blower and HeaterAC states. Blower is an OR state consisting of BlowerOff and BlowerOn states. BlowerOff is an atomic state whereas BlowerOn is an OR state consisting of Blower1 and Blower2 atomic states. HeaterAC is an OR state and consists of atomic states: HeaterACOff, HeaterAct, and ACAct. The entry actions in the states are assignments to output variables of the chart.

The chart is activated by a time-triggered signal shown as System trigger in Figure 1(b) and marked as an event E in the chart in Figure 1(a). The transitions in the chart are enabled by the event E. If the corresponding guard conditions are satisfied then a transition takes place. The function abs returns an absolute value of its argument and the predicate in returns true if its argument state is active.

The input variables of the Heater control block are $x_0\_x_1$ and B and the output variable is $T_H = f(B) \times g(x_0\_x_1)$. Given a value of $B \in \{0.2, 0.4\}$, $f(B)$ evaluates to a constant. The function g is defined as a piecewise linear function using a 1-dimensional lookup table. If (x, y) and (x', y') are two entries of the table and $x_0\_x_1 \in [x, x']$ then $g(x_0\_x_1)$ is defined as the linear interpolation between the entries. The AC control block is similar. We now explain the translation from Simulink models to LHSs with Heater control subsystem as an example. The states and blocks that we consider as corresponding to the subsystem are shaded in Figure 1.

4.2 Translation scheme

Consider a Simulink model $SL = (D, B, C, L)$. For simplicity, we assume that it contains only a single Stateflow chart $SF = (V, E, S, TR, T)$. SL can be systematically translated to an LHS $H = (Q, X, Init, Flow, Jump)$ subject

to certain conditions as explained below. We have used the translation scheme sketched here to derive hybrid systems for the case studies considered by us (Section 5).

Discrete states. The set Q of discrete states can be identified with all possible valuations to the output variables V_O of SF which determine modes of the Simulink model. A Stateflow chart may have integer or real valued output variables and can be considered as auxiliary variables. For instance, the output variable $x_0\_x_1$ of the chart in the VCC model is real valued. Valuations of auxiliary variables of SL that take discrete-values and affect the dynamics of the model however are considered in determining the set Q.

For example, in the VCC model, values of the variables H, A, and B (which are output variables of SF), and values of the row-index of the lookup table (which is an auxiliary variable) determine modes of the model. Thus, a discrete state $q \in Q$ is a 4-tuple (h, a, b, r) where $h, a \in \{0, 1\}$ denote values of variables H and A, $b \in \{0.0, 0.2, 0.4\}$ is a value of the variable B, and r denotes that rows r and $r + 1$ are looked-up. For brevity, let us consider only the first 5 rows of the lookup table of the Heater control block. Thus, the set of discrete states of the Heater control subsystem is $Q = \{q_0 = (0, 0, 0.0, 0), q_1 = (1, 0, 0.2, 1), q_2 = (1, 0, 0.2, 2), q_3 = (1, 0, 0.2, 3), q_4 = (1, 0, 0.2, 4)\}$.

The correspondence between the states of SF and of the hybrid system is determined by the actions of the states in an active configuration of SF. For instance, the active configuration (shaded states) in Figure 1(a) corresponds to the states q_1 to q_4 given above.

Real-valued variables. We identify the set X of real-valued variables from the output variables of integrator, unit delay, or state-space blocks of SL and the input variables D_I. In the VCC model, the Dynamics block has an integrator, defining x_1 as the state variable. The real-valued variables for the model are $X = \{x_0, x_1\}$ where x_0 is the input variable.

Initial states. Let the set of states of SF with default incoming transitions be denoted by a discrete state $q_0 \in Q$. The guards on the default incoming transition(s) and the saturation blocks (if any) associated with input variables determine the set of initial continuous states $Init(q_0)$ which we assume is a convex polyhedron. If a discrete state $q \in Q$ is not an initial state then the convex polyhedron $Init(q) = \emptyset$.

For example, the default incoming transitions in the Stateflow chart in Figure 1(a) are to the states BlowerOff and HeaterACOff. Thus, the initial state is $q_0 = (0, 0, 0.0, 0)$. Note that while Blower1 also has a default incoming transition, Blower itself is an (exclusive) OR state implying that BlowerOn (and hence Blower1) are inactive when BlowerOff is active. The saturation block which follows the user setpoint block in Figure 1(b) determines the range on x_0. For x_1 (the internal temperature) we assume the same range. Therefore, the initial states of the model are:

$$\begin{aligned} Init(q_0) &= 173 \le x_0 \le 373 \wedge 173 \le x_1 \le 373 \\ Init(q_i) &= \emptyset \text{ (for } i \in [1, 4]) \end{aligned}$$

Flow functions. For a discrete state $q \in Q$, we identify the flow expression for a variable $x_i \in X$ by (symbolically) evaluating the data flow path to the input of the integrator or the state-space block, say B_i, of which x_i is an output variable. The input to B_i is the derivative $\dot{x}_i$ of x_i. For a unit delay block, it is the next value of x_i. Note that since a discrete state q determines the control mode completely, all Stateflow dependent choices (outputs of switches, rows of lookup tables, etc.) are resolved. Thus, a data flow path defining the flow function $Flow(q, x_i)$ for a variable x_i at a discrete state q is uniquely identified. We assume that the flows are linear expressions over X. Simulink provides a feature to display the sorted order in which the blocks are executed and is useful for determining the flow functions.

In the hybrid system model of the Heater control subsystem, a discrete state $q = (h = 1, a = 0, b, r)$ determines the indices r and $r + 1$ of the lookup table rows. The output T_H of the Heater control block is defined as $f(b) \times g_{LI}(r, x_0\_x_1)$ where $f(b)$ evaluates to a constant and the function $g_{LI}(r, x_0\_x_1)$ is the linear interpolation of the table entries indexed by r and $r + 1$.

From HeaterAC state, we know that H and A cannot be 1 simultaneously. Thus, the AC control block is disabled and its output T_{AC} is set to 0. Considering that $f_{dy}(In1, In2, In3)$ denotes the expression of the Dynamics block upto the input of the integrator, we get the following flow functions:

$$Flow(q, x_1) = f_{dy}(x_1, f(b) \times g_{LI}(r, x_0 - x_1), 0)$$

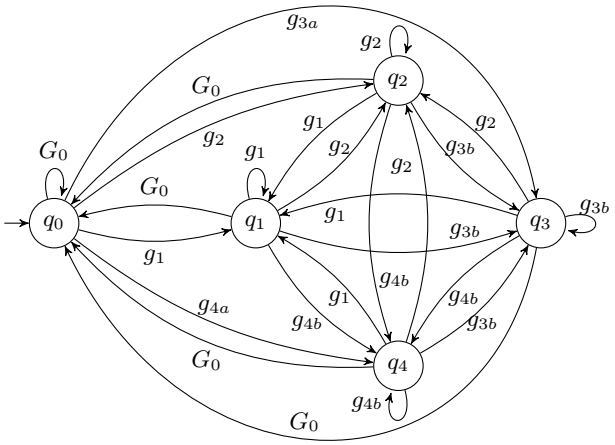
For the input variable x_0, $Flow(q, x_0) = 0$.

Discrete transitions. A transition is taken as a result of an enabling event. An event e can be either a time-triggered event or an asynchronous broadcast event. A time-triggered event affects the sampling time of the model (ref. Section 4.3). For a broadcast event, we can identify the constraints (over the state and input variables of SL or equivalently the real-valued variables X of H) that enable a broadcast of the event. Let these (linear) constraints be given by a formula φ_e. In our examples, we can assume that until an event is completely processed, no other event occurs.

The Stateflow chart SF is evoked upon an occurrence of an input event e. The event is processed downwards from the root of the state hierarchy (S, TR). Suppose a transition $t = (s, s', e, \varphi, act_{condition}, act_{transition}) \in T$ is enabled by e. If the state $s \in S_{Act}$ and the guard φ is true then the transition from s to s' takes place. Let tr be the smallest subtree of TR containing states s and s'. The transition involves marking the states from s to the root of tr as inactive and the states from the root of tr to s' as active such that the resulting active configuration respects the AND/OR nature of composite states (Definition 6). If no such transition is possible then the event is ignored.

Recall that the guards are specified as quantifier-free formulae in predicate logic over the local variables. To ensure that the Simulink model can be formalized as a linear hybrid system, we require that the local variables V_L be defined as linear transformations of the state variables X. Let $\varphi' = \varphi_e \wedge \varphi$ and $\bigvee_i \varphi'_i$ be the disjunctive normal form of φ'. Clearly each disjunct φ'_i is a convex polyhedron.

Each such convex polyhedron φ'_i determines the guard g_i on a transition, say t_i, from q to q' where q and q' correspond respectively to the active configurations before and after the transition t of SF. Let r_i be the reset map for the transition. The reset map r_i is obtained as a sequential composition of (1) $act_{condition}$, (2) the during actions of the active states



$g_{0a} = 2x_0 - 2x_1 \leq 1,\ g_{0b} = x_0 - x_1 \geq 203,\ G_0 = g_{0a} \vee g_{0b}$
$g_1 = 2x_0 - 2x_1 > 1 \wedge x_0 - x_1 \leq 3$
$g_2 = x_0 - x_1 > 3 \wedge x_0 - x_1 \leq 5$
$g_{3a} = x_0 - x_1 > 5 \wedge x_0 - x_1 \leq 10$
$g_{3b} = x_0 - x_1 > 5 \wedge x_0 - x_1 \leq 7$
$g_{4a} = x_0 - x_1 > 10 \wedge x_0 - x_1 \leq 50$
$g_{4b} = x_0 - x_1 > 10 \wedge x_0 - x_1 \leq 12$

Figure 2: Transitions of Heater control subsystem

from (but excluding) the root of tr upto s, (3) the exit actions of the states from s upto the root of tr, (4) $act_{transition}$, and (5) the entry actions of the states from (but excluding) the root of tr to s'. Thus, the transition $t \in T$ is mapped to a set of transitions (q, q', φ_i', r_i) in *Jump*. We model the inner transitions and the steps involving evolution of only continuous states as self-transitions.

Temperature control chart is activated by a time-triggered signal shown as System trigger in Figure 1(b) and marked as an event E in the chart in Figure 1(a). The transitions in the chart are enabled by the event E. In Figure 2, we show the discrete transitions for the Heater control subsystem. The guards are obtained by taking conjunction of the guards on the substates in the Stateflow chart and the lookup table intervals. For instance, the lookup table interval corresponding to q_3 is $(5, 10]$ and this gives the guard g_{3a} for the transition from q_0 to q_3. If the system is in any of the states q_1 to q_4, it can make a transition to q_3 if the guard g_{3b} holds. This is because if Blower1 is active and the temperate difference is in $[5, 7]$ then the model remains in Blower1 (ref. Figure 1(a)) and in conjunction with the lookup table interval $(5, 10]$ we get g_{3b}. There is no reset action with the transitions.

4.3 Simulation time step

Some Simulink blocks have an explicit sampling time while most inherit the sampling time from the blocks connected to their inputs or outputs. The fundamental sampling time is the greatest integer divisor of the sampling times of all the discrete blocks in the model and determines the simulation time step h of the model. During simulation, the outputs of the continuous blocks of the model are computed by a chosen integration routine $\mathbb{S}$ with respect to time step h.

For example, in the VCC model, the System trigger block decides the sampling time of the Stateflow chart and it is propagated to the rest of the model.

4.4 Scope of the translation

While the translation scheme described here covers a commonly used subset of the Simulink language like Stateflow, lookup tables, enabled subsystems, and linear transfer functions; it is not complete. For instance, it does not cover user defined S-Function blocks, variable step integration routines, and zero crossing detection. It also does not cover multirate systems and minor time steps used in some integration routines. Automating the translation scheme is a future work.

5. CASE STUDIES

We have analyzed two case studies: room heating benchmark (RHB) and vehicle climate control (VCC). RHB is a benchmark for hybrid systems verification [12]. VCC is provided as a Simulink demo by Mathworks [1]. VCC uses many Simulink features like lookup tables, enabled subsystems, and concurrency and hierarchy in Stateflow charts. A simplified and partial model of VCC is discussed in Section 4.

Table 1 summarizes the characteristics of the case studies. $\#Q$s and $\#X$s are respectively the number of discrete states and real-valued variables. #Trans. and h are respectively the number of discrete transitions and the value of the simulation time step. In the hybrid system model, the guard on a discrete transition should to be a convex polyhedron. If in the Simulink model a guard is disjunctive then we create a separate transition for each disjunct which results in a high number of transitions. RHB and VCC have approximately over 150 and 10^4 discrete transitions respectively. In the case studies, we use the Euler integration routine.

Case study	$\#Q$s	$\#X$s	#Trans.	h
RHB	12	3	> 150	1/100
VCC	106	2	$> 10^4$	1/120

Table 1: Characteristics of the case studies

5.1 Room heating benchmark

The RHB model considered by us (whose Simulink design is taken from [2]) consists of a house with 3 rooms and 2 heaters. The temperature in a room i is modeled as a real-valued variable x_i. A room can have at most one heater. The set of possible distributions of heaters is $H_D = \{110, 011, 101\}$. For a heater distribution $h \in H_D$, the value at the ith index h_i indicates whether a heater is available in room i ($h_i = 1$ means available). Each distribution is further augmented with four possible heater on/off configurations $Z = \{00, 01, 10, 11\}$. The set of discrete states is $Q = H_D \times Z$. Thus there are total 12 discrete states.

The change in temperature of a room depends on its own temperature, the temperatures of the adjacent rooms, the external temperature u (a constant), and whether a heater is available and on/off. For a discrete state $q = (h, z) \in Q$, the derivative of x_i is defined by the following flow function:

$$\dot{x_i} = \mathit{Flow}(q, x_i) := c_i m_i + b_i(u - x_i) + \sum_{i \neq j} a_{i,j}(x_j - x_i)$$

where $m_i = 1$ if there is a heater in room i ($h_i = 1$) and is on ($z_j = 1$ if it is the jth heater).

In the specific instance of RHB that we analyzed, $b = \langle 0.4, 0.3, 0.4\rangle$, $u = 4$, $c = \langle 6, 7, 8\rangle$, and the values $a_{i,j}$ are given by the following matrix:

$$\begin{pmatrix} 0.0 & 0.5 & 0.0 \\ 0.5 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 \end{pmatrix}$$

The above matrix is symmetric, indicating that two adjacent rooms affect each others' temperature in a symmetric manner. The non-zero values indicate that rooms 1–2 and rooms 2–3 are adjacent. Rooms 1 and 3 are not adjacent.

A heater is moved from a room i to an adjacent room j if (1) room i has a heater ($h_i = 1$), (2) room j does not have a heater ($h_j = 0$), (3) the temperature $x_j \leq get_j$, and (4) $x_i - x_j \geq dif_j$. When two heaters can be moved, the heater from the smaller numbered room is moved. The heater in room i is turned off if $x_i \geq off_i$ and is turned on if $x_i \leq on_i$. For the case study, we considered $get = \langle 18, 18, 18\rangle$, $dif = \langle 1, 1, 1\rangle$, $on = \langle 20, 20, 20\rangle$, and $off = \langle 21, 21, 21\rangle$.

Given two states $q = (h, z)$ and $q' = (h', z')$, the discrete transitions between q and q' and their guards can be identified from the above. Note that the guards should be convex. For instance, when no heater is moved ($h = h'$), we get the following disjunction $x_j > get_i \vee x_i - x_j < dif_j$ leading to a pair of self transitions. We consider the initial state $q_0 = (101, 11)$ where heaters are present in rooms 1 and 3 and are on. The set of initial states $Init(q_0) = 15 \leq x_1 < 21 \wedge 15 \leq x_2 \leq 25 \wedge 15 \leq x_3 < 21$.

5.2 Vehicle climate control

We have discussed a simplified version of VCC in Section 4. We have shown only two substates of BlowerOn in Figure 1(a) whereas the blower output (the variable B) takes a value from $\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ and thus there are 5 substates of BlowerOn in the model. The AC and Heater controls contain 1-dimensional lookup tables with linear interpolation/extrapolation with 7 and 14 rows respectively. The row indices are considered in discrete states as they affect dynamics of the system. $B = 0.0$ iff both AC and Heater are disabled. Further, AC and Heater cannot be active simultaneously. Thus there are 106 discrete states resulting from the distinct values taken by the tuple (H, A, B, R) and over 10^4 discrete transitions. The guards on the transitions are convex and the dynamics in each state is linear.

5.3 Implementation details

Flattening the hierarchical structure of a Stateflow chart leads to a blowup in the number of discrete transitions. We can also observe that guards on several incoming transitions to a state are same (Figure 2). Therefore in our specification format, we model incoming transitions instead of outgoing transitions of a discrete state. This allows us to store a set of transitions with same target state, guard set, and reset map as a single transition. With this representation, we could represent all possible discrete transitions of the VCC model ($> 10^4$) by 657 transitions only. For the RHB model, we could represent all discrete transitions by 128 transitions only. To get simulation trajectories, we add blocks in the models to print the values to the workspace.

We use the Parma Polyhedra Library (PPL) for implementing Algorithm 1. We use the not necessarily closed polyhedra (NNC_Polyhedron) representation for state-sets as some of the guards are strict inequalities. The operations

#Simulations	5	10	15	20
Coverage	83.1%	90.6%	87.4%	89.9%

(a) Room heating benchmark

#Simulations	5	10	15	20
Coverage	17.77%	29.78%	29.49%	30.92%

(b) Vehicle climate control

Table 2: Estimated coverage

PreE and *PreR* are implemented using the affine preimage computations in PPL. To compute preimages when multiple variables are updated simultaneously, we have implemented a preimage computation routine. An appropriate incoming transition (line 3 of Algorithm 1) is identified by iterating over the incoming transitions. We have also implemented an input selection scheme that selects the next input randomly from the as yet uncovered initial state-space.

6. EXPERIMENTAL RESULTS

Coverage of the initial state-space.

The exact computation of coverage of the initial continuous state-space is difficult because the coefficients of the polyhedral constraints that denote the covered region are usually large and the polyhedra can be of arbitrary shapes. Therefore to estimate the coverage of the initial state-space, we select a number of random sample points from the initial state-space and then measure the percentage of the sample points that belong to the covered region.

The continuous state-spaces of RHB and VCC are of 3 and 2 dimensions respectively. Each dimension of the continuous state-space for RHB is in the range $[15, 25]$ and for VCC it is in the range $[173, 373]$. We select a random sampling of 10^3 initial states for RHB and 100^2 states for VCC. The percentage of the sampled states covered by different number of simulations is shown in Table 2.

The length of individual simulations for RHB and VCC was 10 and 100 steps. We chose different initial states at the beginning of each coverage analysis. The number of simulations for a coverage analysis are the column headings in the tables. Our tool randomly selects the initial state for the subsequent simulation from outside the covered region.

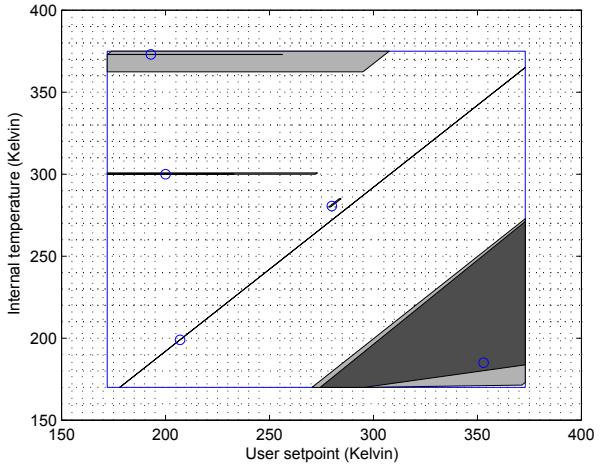
The coverage results demonstrate that a significant coverage of the initial state-space could be obtained in the case studies. For RHB, a set of 5 simulations lead to estimated coverage of 83% and a set of 20 simulations lead to estimated coverage of 90%. For VCC, a set of 5 simulations lead to estimated coverage of 18% and a set of 20 simulations lead to estimated coverage of 31%.

Effect of simulation length on coverage and run-time.

To show the effect of simulation length on coverage, we plot the covered initial state-space for VCC in Figure 3. X-axis shows the user setpoint and Y-axis shows the internal

#Simulations	5	10	15	20
Coverage	35.1%	40.4%	41.3%	43.3%

Table 3: Estimated coverage for RHB for simulations of length 50



Coverage for the VCC model on 5 initial states

Figure 3: Coverage and effect of length of simulation

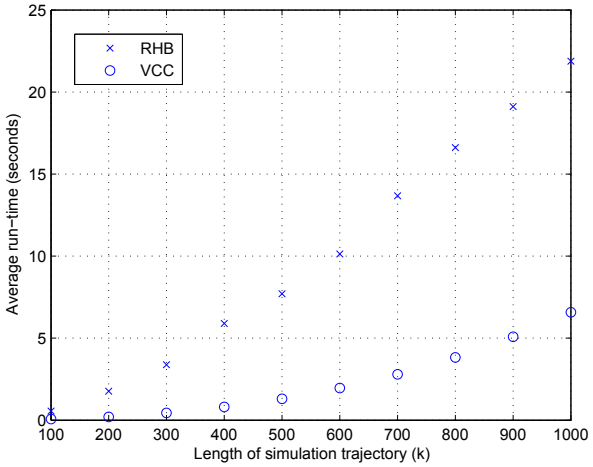


Figure 4: Avg. run-time for diff. simulation lengths

temperature. The initial state-space is a rectangle whose diagonally opposite vertices are $(173, 173)$ and $(373, 373)$. The sample initial states are shown as small circles (centered at them). A simulation is performed starting with each initial state. The corresponding shaded region shows the covered initial state-space.

As the simulation length increases, more constraints need to be satisfied to ensure that the same sequence of discrete states are visited and the equivalence over initial states becomes finer. Thus the coverage decreases with increase in the simulation length. For the top most and bottom most initial states, the coverages shown in lightgray are for simulation length of 100 steps and the coverages shown in darkgray are for simulation length of 1000 steps. For the other states, only covered regions for simulations of 100 steps are shown. The covered regions for simulations of 1000 steps are contained within these regions.

Table 3 gives estimate of coverage for RHB as explained earlier but for simulations of 50 steps and can be compared with the results in Table 2(a).

As the simulation length increases, the run-time of the algorithm also increases. In Figure 4, we plot the average run-time of the symbolic analysis for 5 simulations. We can observe the effects of length of simulation and of the dimension of the continuous state-space on the run-time.

The increase in run-time is a direct consequence of the number of polyhedral computations that need to be performed. With each step of the algorithm, the size of the coefficients of the polyhedral constraints and possibly the number of constraints increase. Our present implementation performs exact polyhedral operations and thus gives reasonable coverage even when simulation lengths are large, allowing deeper explorations of the design as shown in Figure 3 and Table 3. Thus there is a trade-off between the amount of coverage and the run-time.

The dimension of the continuous state-space is known to affect the run-time. It can be seen from the relative run-times for RHB and VCC models (which have respectively 3 and 2 continuous variables). The average time taken for analysis of 1000 step simulations is approximately 6s and 22s for them whereas for 100 step simulations it is small. The run-time is measured on a computer with Intel Core2 T5300 processor (1.73GHz) and Fedora Core 6.

7. CONCLUSIONS

We have presented an analysis technique for linear hybrid systems that integrates numerical simulation with symbolic analysis. We have demonstrated the benefits of the method via two case studies on Simulink/Stateflow models. Our experiments indicate that the symbolic analysis of the numerical simulation from an initial state, typically, allows us to declare a non-trivial region around this initial state to be redundant for future simulations.

The lack of robust tools to map Simulink/Stateflow models to hybrid automata is a hurdle in applying hybrid systems analysis tools to industrially relevant examples. We have reported some progress in this direction, and the VCC example has a number of commonly occurring features. However, an automatic translator from Simulink/Stateflow to hybrid automata is beyond the scope of this paper.

While we have demonstrated that the symbolic analysis yields useful information, it is natural to question whether this analysis would scale. Note that the complexity of the symbolic analysis does not really depend upon the number of discrete states, but depends on the number of simulation steps and the number of continuous variables. Our experiments address increasing the number of simulation steps, but both the examples have small number of continuous variables, and we have not yet applied the tool on other examples. It is well known that the performance of algorithms manipulating polyhedra degrades quickly with increasing dimensions. We believe that simpler representations such as grid and template polyhedra [29], which typically lead to coarse overapproximations in forward symbolic search, can yield effective underapproximations around the concrete simulated execution for our purpose.

Acknowledgments. The work by authors at University of Pennsylvania was partially supported by grants from NSF Cybertrust program and General Motors.

8. REFERENCES

[1] Simulink demos: `http://www.mathworks.com/products/simulink/demos.html`.

[2] Simulink models of hybrid systems benchmarks `http://www.cse.unsw.edu.au/~ansgar/benchmark/`.

[3] A. Agrawal, G. Simon, and G. Karsai. Semantic translation of Simulink/Stateflow models to hybrid automata using graph transformations. *ENTCS*, 109:43–56, 2004.

[4] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[5] R. Alur, T. Dang, and F. Ivancic. Predicate abstraction for reachability analysis of hybrid systems. *ACM Trans. on Embedded Computing Systems*, 5(1):152–199, 2006.

[6] R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 2008.

[7] C. Banphawatthanarak, B.H. Krogh, and K. Butts. Symbolic verification of executable control specifications. In *Intl. Symp. on Computer Aided Control System Design*, pages 581–586. IEEE, 1999.

[8] BEACON Tester, Applied Dynamics International, `http://www.adi.com/products_be_bss_te.htm`.

[9] A. Chutinan and B.K. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *HSCC*, LNCS 1569, pages 76–90. Springer, 1999.

[10] E.M. Clarke, A. Fehnker, Z. Han, B.H. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and counterexample-guided abstraction refinement in model checking of hybrid systems. *Intl. Journ. on Foundations of Computer Science*, 14(4):583–604, 2003.

[11] A. Donzé and O. Maler. Systematic simulation using sensitivity analysis. In *HSCC*, LNCS 4416, pages 174–189. Springer, 2007.

[12] A. Fehnker and F. Ivancic. Benchmarks for hybrid systems verification. In *HSCC*, LNCS 2993, pages 326–341. Springer, 2004.

[13] G. Frehse. Phaver: Algorithmic verification of hybrid systems past HyTech. In *HSCC*, LNCS 3414, pages 258–273. Springer, 2005.

[14] A.A. Gadkari, A. Yeolekar, J. Suresh, S. Ramesh, S. Mohalik, and K.C. Shashidhar. AutoMOTGen: Automatic model oriented test generator for embedded control systems. In *CAV*, LNCS 5123, pages 204–208. Springer, 2008.

[15] A. Girard and G.J. Pappas. Approximation metrics for discrete and continuous systems. *IEEE Trans. on Automatic Control*, 52(5):782–798, 2007.

[16] A. Girard and G.J. Pappas. Verification using simulation. In *HSCC*, LNCS 3927, pages 272–286. Springer, 2006.

[17] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed automated random testing. In *PLDI*, pages 213–223. ACM, 2005.

[18] N. Halbwachs, Y. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *SAS*, LNCS 864, pages 223–237. Springer, 1994.

[19] G. Hamon. A denotational semantics for Stateflow. In *EMSOFT*, pages 164–172. ACM, 2005.

[20] G. Hamon and J.M. Rushby. An operational semantics for stateflow. *STTT*, 9(5-6):447–456, 2007.

[21] D. Harel. Statecharts: A visual formulation for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[22] T.A. Henzinger, P. Ho, and H. Wong-Toi. HyTech: a model checker for hybrid systems. *STTT*, 1, 1997.

[23] A.A. Julius, G.E. Fainekos, M. Anand, I. Lee, and G.J. Pappas. Robust test generation and coverage for hybrid systems. In *HSCC*, LNCS 4416, pages 329–342. Springer, 2007.

[24] A. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *HSCC*, LNCS 1790, pages 202–214. Springer, 2000.

[25] E.A. Lee. What's ahead for embedded software. *IEEE Computer*, pages 18–26, September 2000.

[26] I. Mitchell and C. Tomlin. Level set methods for computation in hybrid systems. In *HSCC*, LNCS 1790, pages 310–323. Springer, 2000.

[27] T. Nahhal and T. Dang. Coverage for continuous and hybrid systems. In *CAV*, LNCS 4590, pages 449–462. Springer, 2007.

[28] Reactis, Reactive Systems, Inc., `http://www.reactive-systems.com`.

[29] S. Sankaranarayanan, T. Dang, and F. Ivancic. Symbolic model checking of hybrid systems using template polyhedra. In *TACAS*, LNCS 4963, pages 188–202. Springer, 2008.

[30] S. Sastry, J. Sztipanovits, R. Bajcsy, and H. Gill. Modeling and design of embedded software. *Proc. of the IEEE*, 91(1), 2003.

[31] M. Satpathy, A. Yeolekar, and S. Ramesh. REDIRECT: Randomized directed testing for Simulink/Stateflow models. In *EMSOFT* (this proceedings). ACM, 2008.

[32] Simulink Design Verifier, The Mathworks, Inc., `http://www.mathworks.com/products/sldesignverifier`.

[33] K. Sen, D. Marinov, and G. Agha. CUTE: a concolic unit testing engine for C. In *FSE*, pages 263–272. ACM, 2005.

[34] Safety Test Builder, TNI-Software., `http://www.tni-software.com/en/produits/safetytestbuilder`.

[35] A. Tiwari. Abstractions for hybrid systems. *Formal Methods in System Design*, 32(1):57–83, 2008.

[36] A. Tiwari. Formal semantics and analysis methods for Simulink Stateflow models. Technical report, SRI International, 2002.

[37] S. Tripakis, C. Sofronis, P. Caspi, and A. Curic. Translating discrete-time Simulink to Lustre. *ACM Trans. on Embedded Computing Systems*, 4(4):779–818, 2005.

[38] T-VEC Tester, T-VEC Technologies, Inc., `http://www.t-vec.com/solutions/simulink.php`.