Synthesizing Switching Logic for Safety and Dwell-Time Requirements

Susmit Jha
UC Berkeley
jha@eecs.berkeley.edu

Sumit Gulwani
Microsoft Research
sumitg@microsoft.com

Sanjit A. Seshia
UC Berkeley
sseshia@eecs.berkeley.edu

Ashish Tiwari
SRI International
tiwari@csl.sri.com

ABSTRACT

Cyber-physical systems (CPS) can be usefully modeled as hybrid automata combining the physical dynamics within modes with discrete switching behavior between modes. CPS designs must satisfy safety and performance requirements. While the dynamics within each mode is usually defined by the physical plant, the tricky design problem often involves getting the switching logic right. In this paper, we present a new approach to assist designers by synthesizing the switching logic, given a partial system model, using a combination of fixpoint computation, numerical simulation, and machine learning. Our technique begins with an over-approximation of the guards on transitions between modes. In successive iterations, the over-approximations are refined by eliminating points that will cause the system to reach unsafe states, and such refinement is performed using numerical simulation and machine learning. In addition to safety requirements, we synthesize models to satisfy dwell-time constraints, which impose upper and/or lower bounds on the amount of time spent within a mode. We demonstrate using case studies that our technique quickly generates intuitive system models and that dwell-time constraints can help to tune the performance of a design.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; I.2.2 [**Artificial Intelligence**]: Program Synthesis; K.3.2 [**Learning**]: Concept Learning

General Terms

Design, Verification, Performance

Keywords

Automated synthesis, Oracle-based learning, Switching logic, Cyber-physical Systems, Hybrid Systems

1. INTRODUCTION

As cyber-physical systems (CPS) are increasingly deployed in transportation, health-care, and other societal-scale applications, there is a pressing need for automated tool support to ensure dependability while enabling designers to meet shortening time-to-market constraints. Model-based design tools enable designers to work at a high level of abstraction, but there is still a need to assist the designer in creating *correct* and *efficient* systems.

A holy grail for the design of CPS is to automatically synthesize models from safety and performance specifications. In its most general form, automated synthesis is very difficult to achieve, in part because synthesis often involves human insight and intuition, and in part because of system complexity – the tight integration of complex continuous dynamics with discrete switching behavior can be tricky to get correct. Nevertheless, in some contexts, it may be possible for automated tools to complete partial designs generated by a human designer, thus enabling the designer to efficiently explore the space of design choices whilst ensuring that the synthesized system remains safe.

In this paper, we consider a special class of synthesis problems, namely synthesis of mode switching logic for multi-modal dynamical systems (MDS). An MDS is a physical system (plant) that can operate in different modes. The dynamics of the plant in each mode is known. However, to achieve safe and efficient operation, it is often necessary to switch between the different operating modes. Designing correct switching logic can be tricky and tedious. We consider the problem of automatically synthesizing switching logic, given the intra-mode dynamics, so as to preserve safety in MDS. The human designer can guide the synthesis process by providing initial approximations of the switching guards and a library of expressions (components) using which the guards can be synthesized.

Our synthesis approach performs reasoning within each mode and reasoning across modes in two different ways. Within each mode, reasoning is based entirely on using *numerical simulations*. While this can lead to potential unsoundness, it allows us to handle complex and nonlinear dynamics that are difficult to reason about in any other way. Across modes, reasoning is performed using *fixpoint computation techniques*. Similar to abstract interpretation, computation of the fixpoint is performed over an "abstract domain," which is specified by the user in the form of a component library for the switching guards. Each step of the fixpoint computation involves the use of *machine learning* to learn improved approximations of the switching guards based on the results of numerical simulations.

The key contribution of our paper is a *new approach for synthesizing safe switching logic based on integrating numerical simulation, machine learning, and fixpoint iterations*. In addition to safety, our approach also extends to handling dwell-time requirements, which impose upper and/or lower bounds on the amount of time spent within a mode. While

ICCPS '10, April 13-15, 2010, Stockholm, Germany.

numerical simulations have been used to perform formal verification (e.g., [8, 5, 4]), to our knowledge our approach is the first to use simulations to perform synthesis with safety guarantees. We demonstrate using case studies (Sec. 3 and 6) that our technique generates intuitive system models and that dwell-time constraints can help to tune the performance of a design.

2. PROBLEM DEFINITION

In this section, we describe the problem of synthesizing switching logic for a multi-modal continuous dynamical system. We present two versions of the problem. In the first version, we ask for a switching logic that only preserves safety. In the second version, we also require that the synthesized system satisfy some dwell-time requirements in each mode. We begin with some definitions.

A *continuous dynamical system* (CDS) is a tuple $\langle X, f\rangle$ where X is a finite set of $|X| = n$ real-valued variables that define the state space $\mathbb{R}^n$ of the continuous dynamical system, and $f : \mathbb{R}^n \mapsto \mathbb{R}^n$ is a vector field that specifies the continuous dynamics as $\frac{d\mathbf{x}}{dt} = f(\mathbf{x})$. The vector field f is assumed to be locally Lipschitz at all points, which guarantees the existence and uniqueness of solutions to the ordinary differential equations.

Often, a system has multiple modes and in each mode, its dynamics is different. Such a multi-modal system behaves as a different continuous dynamical system in each mode.

Definition 1 (Multi-modal CDS (MDS)). *An MDS is a tuple $\langle X, I, f_1, f_2, \ldots, f_k\rangle$ where*

- *$\langle X, f_i\rangle$ is a continuous dynamical system (representing the i-th mode)*
- *$I \subseteq \mathbb{R}^n$ is the set of initial states*

We will use $M = \{1, 2, \ldots, k\}$ as the set of indices of the modes. A *trajectory* for MDS is a continuous function $\tau(t) : [0, \infty) \mapsto \mathbb{R}^n$ if there is an increasing sequence $t_0 := 0 < t_1 < t_2 \ldots$ such that

- $\tau(0) \in I$,
- for each interval $[t_i, t_{i+1})$, there is some mode $j \in M$ such that $\frac{d\tau}{dt}(t) = f_j(\tau(t))$ for all $t_i \leq t < t_{i+1}$, and
- $j = 1$ when $t_i = 0$ (that is, we start in Mode 1.).

A multi-modal system can nondeterministically switch between its modes. The goal is to control the switching between different modes to achieve safe operation.

Definition 2 (Switching logic (S)). *A switching logic S for an MDS $\langle X, I, (f_i)_{i\in M}\rangle$ is a tuple $\langle (g_{ij})_{i\neq j; i,j\in M}\rangle$, containing guards $g_{ij} \subseteq \mathbb{R}^n$.*

A multi-modal system MDS can be combined with a switching logic S to create a hybrid system $\texttt{HS} := (\texttt{MDS}, S)$ in the following natural way: the hybrid system HS has k modes with dynamics given by $\frac{dX}{dt} = f_i$ in mode i, and with g_{ij} being the guard on the discrete transition from mode i to mode j. The initial states of HS are I in Mode 1, where I is the set of initial states of the MDS. The discrete transitions in HS have identity reset maps, that is, the continuous variables do not change values during discrete jumps. The state invariant Inv_i for a mode $i \in I$ is the (topological) closure of the complement of the union of all guards on outgoing transitions; in other words $Inv_i := Closure(\mathbb{R}^n - \bigcup_{j\in I} g_{ij})$. Note that we are assuming here that a discrete transition is taken as soon as it is enabled.[1] This completes the definition of the hybrid system. The semantics of hybrid systems that defines the set of *reachable states* of hybrid systems is standard [1].

A safety property is a set $\phi_S \subseteq \mathbb{R}^n$ of states. We will overload ϕ_S to also denote the predicate $\phi_S(X)$. A state $\mathbf{x}$ is said to be *safe* if and only if $\mathbf{x} \in \phi_S$ (or equivalently, if $\phi_S(\mathbf{x})$ is true). A hybrid system HS is safe with respect to ϕ_S if and only if all the reachable states in HS are safe.

Coming up with the correct guards for the mode switches such that all reachable states are safe is challenging and our proposed technique aims at automating this task. While controller synthesis has been widely studied, what differentiates our work is that we provide the designer an option to provide some initial partial design. Specifically, we assume that the designer can provide an over-approximations for the guards. In the extreme case, if transition from mode i to mode j is disallowed, then the designer can set $g_{ij} = \emptyset$, and if the designer knows nothing about the possibility of a transition from mode i to mode j, then she can set $g_{ij} = \mathbb{R}^n$. The designer can specify partial information by picking an intermediate set as the initial guard.

If $S := \langle (g_{ij})_{i,j\in M}\rangle$ and $S' := \langle (g'_{ij})_{i,j\in M}\rangle$ are two switching logics, then we use the notation $S' \subseteq S$ to denote that $g'_{ij} \subseteq g_{ij}$ for all $i, j \in M$.

We provide two variants of the problem definition.

Definition 3 (Switching logic synthesis problem v1). *Given a multi-modal continuous dynamical system (MDS), a switching logic S, and the safety specification ϕ_S, the switching logic synthesis problem seeks to synthesize a new switching logic S' such that*
(1) $S' \subseteq S$ and
(2) the hybrid system $\texttt{HS} := (\texttt{MDS}, S')$ is safe with respect to ϕ_S.

Consider the case when the designer provides no information and sets all guards to $\mathbb{R}^n$. In this case, it is trivial to synthesize a safe hybrid system by just setting all switching guards to be ϕ_S. The reader can check that this is a solution for the switching logic synthesis problem defined above. This solution is, however, undesirable since the resulting hybrid system has only *zeno behaviors*, i.e., an infinite number of transitions can be made in finite time (as we are assuming that a transition is taken as soon as it is enabled).

The second problem definition below gives the designer a way to explicitly rule out solutions that have zeno behavious. Specifically, the user can specify (both lower and upper) bounds on the amount of time every trajectory should spend in a mode.

Definition 4 (Switching logic synthesis problem v2). *Given a multi-modal continuous dynamical system (MDS), a switching logic S, a sequence $\langle te_1, \ldots, te_k\rangle$ of non-negative minimum-dwell time requirements, a sequence $\langle tx_1, \ldots, tx_k\rangle$ of non-negative maximum-dwell time requirements, and a safety specification ϕ_S, the switching logic synthesis problem seeks to synthesize a new switching logic S' such that*
(1) $S' \subseteq S$,
(2) the hybrid system $\texttt{HS} := (\texttt{MDS}, S')$ is safe with respect to ϕ_S, and
(3) whenever any trajectory of HS enters mode i, it stays in mode i for atleast te_i and atmost tx_i time units.

[1] Assume that the mode dynamics are not tangential to the state invariant at any point.

The designer can now force the synthesis of only nonzeno systems by setting te_i to a strict positive number for selected modes. Note that if the designer sets te_i to zero and tx_i to ∞ for all modes, then the second problem is the same as the first problem.

Notation

Our paper makes use of the formal definitions of temporal formulas and the evaluation of a temporal formula in a given dynamical system as given below.

Consider the weak until $\mathbf{W}$ and the strong until $\mathbf{U}$ temporal logic operators. Recall that we do not distinguish between a set of states and a predicate on states. A *state formula* is a predicate on states or a Boolean combination of predicates. If ϕ, ϕ' are sets of states, then $\phi \mathbf{W} \phi'$ and $\phi \mathbf{U} \phi'$ are *temporal formulas*.

A state formula is evaluated over a state. The formula ϕ evaluates to true on a state $\mathbf{x}$ if $\mathbf{x} \in \phi$. A temporal formula is evaluated over a given trajectory τ. The formula $\phi \mathbf{U} \phi'$ evaluates to true on trajectory τ if

$$\exists t_0 : \tau(t_0) \in \phi' \ \wedge \ (\forall 0 \leq t < t_0 : \tau(t) \in \phi) \tag{1}$$

Informally, the temporal formula $\phi \mathbf{U} \phi'$ is true if ϕ' becomes true eventually and until it becomes true, ϕ is true. The weak until operator, $\mathbf{W}$, is a weaker notion and does not require that ϕ' necessarily becomes true. If ϕ, ϕ' are sets of states, then the temporal formula $\phi \mathbf{W} \phi'$ evaluates to true over a given trajectory τ if

$$(\exists t_0 : \tau(t_0) \in \phi' \ \wedge \ (\forall 0 \leq t < t_0 : \tau(t) \in \phi)) \ \vee \\ (\forall t \geq 0 : \tau(t) \in \phi) \tag{2}$$

For uniformity, a state formula can be evaluated on a trajectory as follows: a state formula ϕ evaluates to true on a trajectory τ if $\tau(0) \in \phi$. We can combine state and temporal formulas using Boolean connectives and evaluate them over trajectories using the natural interpretation of the Boolean connectives. If Φ is a state or temporal formula, then we write

$$Mode_i, \ I \ \models \ \Phi$$

to denote that the formula Φ evaluates to true on *all* trajectories of the CDS in mode i that start from a state in I.

3. OVERVIEW

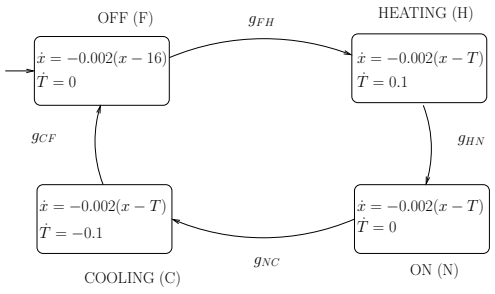

Figure 1: Thermostat

In this section, we present an overview of our approach using a thermostat controller [10] as an example. The 4-mode thermostat controller is presented in Figure 1. The room temperature is represented by x and the temperature of the heater is represented by T. The initial condition I is given by $T = 20^\circ$C and $x = 19^\circ$C. The safety requirement ϕ_S is that the room temperature lies between 18°C and 20°C, that is, ϕ_S is $18 \leq x \leq 20$. (We omit the units in the sequel, for brevity.)

In the OFF mode, the temperature falls at a rate proportional to the difference between the room temperature x and the temperature outside the room which is assumed to be constant at 16. In the HEATING mode, the heater heats up from 20 to 22 and in the COOLING mode, the heater cools down from 22 to 20. In the ON mode, the heater is at a constant temperature of 22. In the HEATING, ON and COOLING mode, the temperature of the room changes in proportion to the difference between the room temperature and the heater temperature. We need to synthesize the four guards: g_{FH}, g_{HN}, g_{NC} and g_{CF}.

The guards must respect the safety property on the room temperature x as well as the specification on the heater temperature T in HEATING and COOLING mode. So, from the given specifications, we know that

$$\begin{aligned} g_{FH} &\subseteq 18 \leq x \leq 20 \wedge T = 20 \\ g_{HN} &\subseteq 18 \leq x \leq 20 \wedge T = 22 \\ g_{NC} &\subseteq 18 \leq x \leq 20 \wedge T = 22 \\ g_{CF} &\subseteq 18 \leq x \leq 20 \wedge T = 20 \end{aligned} \tag{3}$$

In order that the MDS remains safe, we need to ensure that all states reachable within each mode are safe. Consider the OFF mode. We need to ensure that all traces starting from some point in the initial condition I or g_{CF} do not reach an unsafe state before reaching some state in g_{FH}. Reaching some state in g_{FH} enables a transition out of the OFF mode. In other words, the first two temporal properties in Equation 4 must be satisfied by all traces in the OFF mode. Similarly, for HEATING mode, all traces starting from some state in $x \in g_{FH}$ must not reach an unsafe state before reaching an exit state in g_{HN}, as indicated by the third property below. For the other two modes, similar temporal properties on the traces need to be enforced. Overall, the following temporal assertions can be written for the four guards.

$$\begin{aligned} F, I &\models \phi_S \ \mathbf{W} \ g_{FH} \\ F, g_{CF} &\models \phi_S \ \mathbf{W} \ g_{FH} \\ H, g_{FH} &\models \phi_S \ \mathbf{W} \ g_{HN} \\ N, g_{HN} &\models \phi_S \ \mathbf{W} \ g_{NC} \\ C, g_{NC} &\models \phi_S \ \mathbf{W} \ g_{CF} \end{aligned} \tag{4}$$

Switching Logic Synthesis Problem v1: We can synthesize a safe switching logic by computing the fixpoint of the above 5 assertions in Equation 4. We initialize using the equations in Equation 3 obtained from the safety and other user provided specifications which put an upper bound on the guards. We then perform a *greatest fixpoint computation*: in each iteration, we remove states from the guards which would lead to some unsafe state in a mode. Fixpoint computation leads to the following guards which ensure that all states reachable are safe. We compute only till the second place of decimal.

$$\begin{aligned} g_{FH} &: 18.00 \leq x \leq 19.90 \wedge T = 20 \\ g_{HN} &: 18.00 \leq x \leq 19.95 \wedge T = 22 \\ g_{NC} &: 18.00 \leq x \leq 19.95 \wedge T = 22 \\ g_{CF} &: 18.00 \leq x \leq 20.00 \wedge T = 20 \end{aligned}$$

The behavior of the synthesized thermostat for the first 1000 seconds from the initial state is shown in Figure 2. The room temperature gradually rises from its initial value of 19 and then stays between 19.90 and 20.

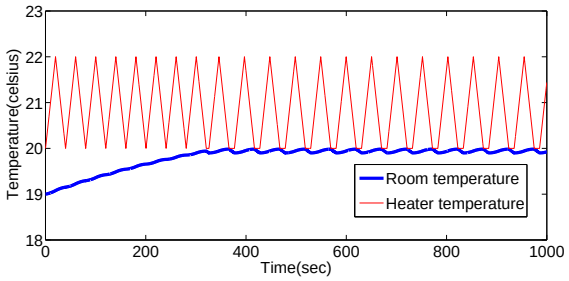

Figure 2: Behavior of Synthesized Thermostat

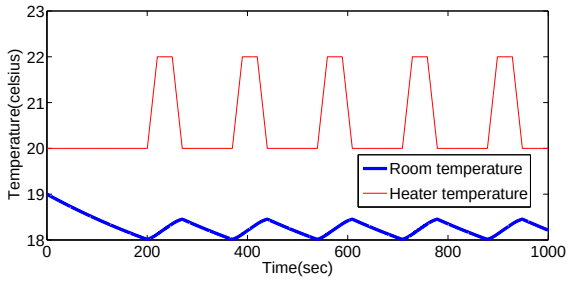

Figure 3: Behavior of Synthesized Thermostat with Dwell Time Specification: Minimum dwell time of 100s in OFF mode.

Switching Logic Synthesis Problem v2: Though the system synthesized above satisfies the safety specification, it has the undesirable behavior of switching frequently. It keeps the room temperature in the narrow interval of $19.90 \leq x \leq 20$, even though the safety condition only required it to be in $18 \leq x \leq 20$. Ideally, designers are interested not only in safe systems but in systems with good performance. The dwell time specification provides a mechanism to the designer to guide our synthesis technique to solutions with good performance.

Minimum dwell-time of 100 seconds in OFF mode (case A): We add an extra constraint in the specification of our synthesis problem that the system must spend atleast 100 seconds in the OFF mode. This would lead to less frequent switching as well as minimize energy consumption since heater remains off in the OFF mode.

Let us add a timer variable t with dynamics $\dot{t} = 1$ in every mode. Assume that t is reset to 0 during every discrete transition. To enforce the minimum dwell-time, the following constraint must also be satisfied in addition to the fixpoint constraints in Equation 4.

$$\begin{aligned} F, I &\models \phi_S \ \mathbf{W} \ (g_{FH} \wedge t \geq 100) \\ F, g_{CF} &\models \phi_S \ \mathbf{W} \ (g_{FH} \wedge t \geq 100) \end{aligned} \tag{5}$$

The guards obtained by computing the fixpoint of equations in (4) and (5) are as follows.

$$\begin{aligned} g_{FH} &: 18.00 \leq x \leq 19.90 \wedge T = 20 \wedge t \geq 100 \\ g_{HN} &: 18.00 \leq x \leq 19.95 \wedge T = 22 \\ g_{NC} &: 18.35 \leq x \leq 19.95 \wedge T = 22 \\ g_{CF} &: 18.45 \leq x \leq 20.00 \wedge T = 20 \end{aligned}$$

Since t was a timer variable we had introduced, we next eliminate it from g_{FH}. We do so by removing states from g_{FH} which are reachable from any state in g_{CF} in less than 100 seconds. These set of states are $18.01 < x \leq 20 \wedge T = 20$. Hence, the final guards that respect the safety property as well as enforce a minimum dwell-time of 100 seconds in OFF mode are as follows.

$$\begin{aligned} g_{FH} &: 18.00 \leq x \leq 18.01 \wedge T = 20 \\ g_{HN} &: 18.00 \leq x \leq 19.95 \wedge T = 22 \\ g_{NC} &: 18.00 \leq x \leq 19.95 \wedge T = 22 \\ g_{CF} &: 18.00 \leq x \leq 20.00 \wedge T = 20 \end{aligned}$$

The behavior of the synthesized thermostat for the first 1000 seconds from the initial state is shown in Figure 3. We observe that the number of switches has gone down from 21 to 5 and the room temperature now stays between 18.01 and 18.45.

Minimum dwell-time of 300 seconds in both OFF and ON mode (case B): We observe that the design synthesized with minimum dwell-time of 100 seconds in OFF mode has relatively less switching but still, we would like to reduce its switching frequency. Also, the room temperature can safely lie between 18 and 20 but in the above synthesized system, it is restricted to a narrow interval of 18.01 and 18.45. So, we increase the minimum dwell-time in OFF mode to 300 seconds. We also enforce a minimum dwell-time of 300 seconds in ON mode to ensure room heats up to a higher temperature within the safe interval.

We now get the following fixpoint equations.

$$\begin{aligned} F, I &\models \phi_S \ \mathbf{W} \ g_{FH} \wedge (t \geq 300) \\ F, g_{CF} &\models \phi_S \ \mathbf{W} \ g_{FH} \wedge (t \geq 300) \\ H, g_{FH} &\models \phi_S \ \mathbf{W} \ g_{HN} \\ N, g_{HN} &\models \phi_S \ \mathbf{W} \ g_{NC} \wedge (t \geq 300) \\ C, g_{NC} &\models \phi_S \ \mathbf{W} \ g_{CF} \end{aligned}$$

Fixpoint computation yields the following guards.

$$\begin{aligned} g_{FH} &: 18.00 \leq x \leq 18.14 \wedge T = 20 \wedge t \geq 300 \\ g_{HN} &: 18.00 \leq x \leq 18.26 \wedge T = 22 \\ g_{NC} &: 19.60 \leq x \leq 19.95 \wedge T = 22 \wedge t \geq 300 \\ g_{CF} &: 19.65 \leq x \leq 20.00 \wedge T = 20 \end{aligned}$$

We restrict g_{NC} and g_{FH} in the same way as (Case A) by computing the set of states reachable from g_{HN} and g_{CF} in less than 300 seconds respectively. The final synthesized guards are as follows.

$$\begin{aligned} g_{FH} &: 18.00 \leq x \leq 18.01 \wedge T = 20 \\ g_{HN} &: 18.00 \leq x \leq 18.26 \wedge T = 22 \\ g_{NC} &: 19.94 \leq x \leq 19.95 \wedge T = 22 \\ g_{CF} &: 19.65 \leq x \leq 20.00 \wedge T = 20 \end{aligned}$$

The behavior of the synthesized thermostat for the first 1000 seconds from the initial state is shown in Figure 4. We observe that the number of switches has gone down to 1 and the room temperature is still within the safe interval of 18 and 20. This example shows how our synthesis approach can be used to synthesize not only safe systems but also systems with desired performance. Dwell-time properties can be used by the user to explore designs with better performance.

4. FIXPOINT ALGORITHM

We are now ready to describe the procedure for solving the switching logic synthesis problem in Definition 3.

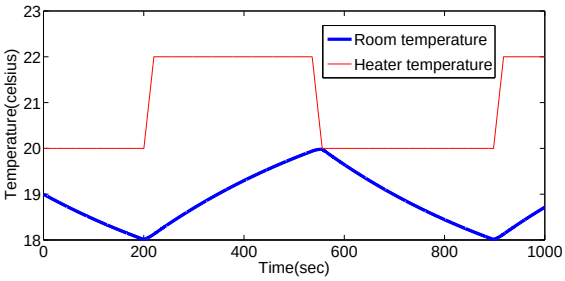

Figure 4: Behavior of Synthesized Thermostat with Dwell Time Specification: Minimum dwell time of 300s in OFF and ON modes.

Assume that we are given an MDS MDS, a safety property ϕ_S, and an over-approximation of the guards S.

$$\texttt{MDS} := \langle X, I, f_1, \ldots, f_k \rangle, \quad \phi_S \subseteq \mathbb{R}^n, \quad S := \langle (g_{ij})_{i,j \in M} \rangle$$

We wish to solve the problem in Def. 3 for these inputs.

Let us say we find guards g'_{ij}'s such that they have the following property: for every mode i, if a trajectory enters mode i (via any of the incoming transitions with guard g'_{ji}), then it remains safe *until* one of the exit guards g_{ik} becomes true. This property can be written formally using the *weak until* operator.

$$\begin{aligned} Mode_1,\ I \quad &\models \quad \phi_S \mathbf{W} (\bigvee_{k \in M} g'_{1k}) \\ Mode_i,\ \bigvee_{j \in M} g'_{ji} \quad &\models \quad \phi_S \mathbf{W} (\bigvee_{k \in M} g'_{ik}) \quad \text{for } i = 1..k \end{aligned} \tag{6}$$

If the guards in the switching logic S' satisfy the collection of assertions in Equation 6, then the resulting hybrid system is safe. The converse is also true.

LEMMA 1. *Given an* $\texttt{MDS} := \langle X, I, f_1, \ldots, f_k \rangle$, *and a safety property* ϕ_S, *if* $S' = \langle (g'_{ij})_{i,j \in M} \rangle$ *is a switching logic that satisfies all assertions in Equation 6, then the hybrid system* $\texttt{HS} := (\texttt{MDS}, S')$ *is safe with respect to* ϕ_S.

Conversely, if there exists a switching logic S' *such that the hybrid system* $\texttt{HS} := (\texttt{MDS}, S')$ *is safe with respect to* ϕ_S, *then there is a switching logic* $S'' \subseteq S'$ *that satisfies the assertions in Equation 6.*

PROOF. The first part follows directly from the definition of the semantics of the temporal operators and our assumption that discrete transitions are taken as soon as they are enabled.

For the converse part, the desired $S'' := \langle (g''_{ij})_{i,j \in M} \rangle$ is obtained by intersecting the set *Reach* of reachable states of HS with S'; that is, $g''_{ij} := g'_{ij} \cap Reach$. The reader can verify that S'' will satisfy the assertions in Equation 6. □

At a semantic level, we can solve the problem in Definition 3 by computing the fixpoint of the assertions in Equation 6. This procedure is presented in Figure 5. The fixpoint iterations start by picking the most liberal guards possible (which is the intersection of the safety property and the user-specified bounds). In each successive step, the guards are made smaller by removing certain "bad" states. Specifically, we remove from g'_{ji} any state that reaches an unsafe state following the dynamics of Mode i, *before* it reaches any

SWITCHSYN1(MDS, ϕ_S, S):

```
1  // Input MDS := ⟨X, I, f_1, ..., f_k⟩,
2  // Input φ_S ⊆ R^n,
3  // Input S := ⟨(g_ij)_{i,j∈M}⟩,
4  // Output synthesis successful/failed
5  for all i,j ∈ M do g'_ij := g_ij ∩ φ_S
6  repeat {
7   for all i ∈ M do {
8     bad := {x | Mode_i, {x} ⊨ ¬(⋁_k g'_ik) U ¬φ_S}
9     for all j ∈ M do g'_ji := g'_ji − bad
10    if (i == 1 and I ∩ bad ≠ ∅)
11       return "synthesis failed"
12  }
13 } until (g'_ij's do not change)
14 if (I ⇒ φ_S)
15    return "synthesis successful"
16 else return "synthesis failed"
```

Figure 5: Procedure for solving the switching logic synthesis problem v1.

exit guard. Thus, in each iteration, we reason locally about only one mode at a time. We stop when we reach a fixpoint.

We state the soundness and completenss of the fixpoint algorithm for solving the switching logic synthesis problem.

THEOREM 1 (SOUNDNESS OF PROCEDURE SWITCHSYN1). *If Procedure* SWITCHSYN1 *terminates with "synthesis successful" and* g'_{ij} *are the discovered guards, then the hybrid system* $\texttt{HS} := (\texttt{MDS}, \langle (g'_{ij})_{i,j \in M} \rangle)$ *is safe for* ϕ_S.

PROOF. (Sketch) If Procedure SWITCHSYN1 terminates with "synthesis successful" and g'_{ij} are the discovered guards, then these guards satisfy all the assertions in Equation 6. Using Lemma 1, we conclude that the hybrid system $\texttt{HS} := (\texttt{MDS}, S')$ is safe with respect to ϕ_S. □

Even when it terminates with success, note that the Procedure SWITCHSYN1 does not guarantee that the synthesized hybrid system HS has nonzeno behaviors. In a post-processing step, one can perform sufficient checks to guarantee the absence of zeno behaviors.

We can also show that our procedure is complete.

THEOREM 2 (COMPLETENESS OF PROCEDURE SWITCHSYN1). *If Procedure* SWITCHSYN1 *terminates with "synthesis failed", then there is no* $S' \subseteq S$ *such that the hybrid system* $\texttt{HS} := (\texttt{MDS}, S')$ *is safe.*

PROOF. Assume that the claim is false and there is a switching logic $S' \subseteq S$ such that $\texttt{HS} := (\texttt{MDS}, S')$ is safe. By Lemma 1, there is a switching logic $S'' := Reach \cap S'$ that satisfies Equation 6. Recall that *Reach* is the set of reachable states of HS. Let $S'_i, i = 0, 1, \ldots$, be the intermediate switching logics computed by Procedure SWITCHSYN1. Clearly, $S'_0 \supseteq S'_1 \supseteq S'_2 \supseteq \cdots$ and $S'_0 := S \cap \phi_S$. Since $Reach \subseteq \phi_S$ by assumption, we can easily verify that $S'' \subseteq S'_0$. We will inductively show that $S'' \subseteq S'_i$ for all i.

Suppose $S'' \subseteq S'_N$. Suppose we go from S'_N to S'_{N+1} by deleting the set *bad* from g'_{ji}. We need to show that $S'' \subseteq$

S'_{N+1}. Let $S'_N := \langle (g'_{Nij})_{i,j\in M}\rangle$ and let $S'' := \langle (g''_{ij})_{i,j\in M}\rangle$.

$\mathbf{x} \in bad$

$\Rightarrow \quad Mode_i, \{\mathbf{x}\} \models \neg(\bigvee_k g'_{Nik})\mathbf{U}\neg\phi_S$

$\Rightarrow \quad Mode_i, \{\mathbf{x}\} \models \neg(\bigvee_k g''_{ik})\mathbf{U}\neg\phi_S, \because g''_{ik} \subseteq g'_{Nik}$

$\Rightarrow \quad Mode_i, \{\mathbf{x}\} \models \neg(\phi_S \mathbf{W} \bigvee_k g''_{ik})$

$\Rightarrow \quad Mode_i, \{\mathbf{x}\} \not\models \phi_S \mathbf{W} \bigvee_k g''_{ik}$

$\Rightarrow \quad \mathbf{x} \notin g''_{ji}, \ \because S''$ satisfies Equation 6

$\mathbf{x} \notin I$ if $i == 1, \because S''$ satisfies Equation 6

This shows that $S'' \subseteq S'_{N+1}$ and Procedure SwitchSyn1 cannot return at Line 11. Since HS is assumed to be safe, $I \Rightarrow \phi_S$ and hence Procedure SwitchSyn1 cannot return at Line 16. Hence, Procedure SwitchSyn1 can only return "synthesis successful" contradicting our assumption. □

4.1 Switching Logic Synthesis V2

We now consider the switching logic synthesis problem in Definition 4. Recall that apart from the bounds on the guards, the user can provide minimum and maximum dwell time requirements for each mode. The goal is to synthesize a switching logic where the guards satisfy the specified bounds and the trajectories of the resulting hybrid system satisfy the minimum and maximum dwell time requirements.

Procedure SwitchSyn2 for solving the problem in Definition 4 is outlined in Figure 6. Procedure SwitchSyn2 runs in three phases. In the first step, the new problem is transformed to the old problem. In the second step, Procedure SwitchSyn1 is used to solve the generated problem. In the third step, the result is transformed back to get a result of the given problem.

Suppose that we are given

$\texttt{MDS} := \langle X, I, f_1, \ldots, f_k\rangle, \ \ \phi_S \subseteq \mathbb{R}^n, \ \ S := \langle (g_{ij})_{i,j\in M}\rangle,$
$\texttt{Te} := \{te_1, \ldots, te_k\}, \ \ \texttt{Tx} := \{tx_1, \ldots, tx_k\}$

In the first step, the problem in Definition 4 is reduced to the previous problem. This reduction is achieved by introducing a new state variable t such that
(1) the dynamics of t is given by $\dot{t} = 1$ in each mode
(2) the variable t is reset to 0 in each discrete transition
These two steps are performed by the function `Add_timer_t`. Now, the dwell time requirements can be specified as bounds on the variable t. Specifically, the over-approximation S of the guards can be updated as follows:

$$g_{ij} \quad := \quad g_{ij} \wedge (te_i \leq t \leq tx_i)$$

In the second step, a call to Procedure SwitchSyn1 is made, but with the updated S. Recall that Procedure SwitchSyn1 essentially performs an iterative fixpoint computation to solve Equation 6. Equation 6 assumes that discrete transitions do not reset any continuous variables. Since we now have discrete transitions that reset t to 0, we need a slightly modified Procedure SwitchSyn1 that solves the modified equations below:

$$Mode_1, R(I) \ \models \ \phi_S \mathbf{W} \bigvee_{k\in M} g'_{1k}$$

$$Mode_i, \bigvee_{j\in M} R(g'_{ji}) \ \models \ \phi_S \mathbf{W} \bigvee_{k\in M} g'_{ik} \ \text{ for } i = 1..k \qquad (7)$$

```
SwitchSyn2(MDS, φ_S, S, Te, Tx):
1  // Input MDS, φ_S, S:  As in Figure 5
2  // Input Te := ⟨te_1, ..., te_k⟩
3  // Input Tx := ⟨tx_1, ..., tx_k⟩
4  // Output synth.  successful/failed
5  MDS^e  := Add_timer_t(MDS)
6  S^e  := ⟨(g_ij ∧ (te_i ≤ t ≤ tx_i))_{i,j∈M}⟩
7  // Call SwitchSyn1 with the updated S
8  res  := SwitchSyn1(MDS^e, φ_S, S^e)
9  if res == "synthesis failed"
10    return "synthesis failed"
11 else let S' be the synthesized guards
12 // post processing step
13 for all i, j ∈ M do
14  g_ij := {x | ⟨x, t⟩ ∈ g'_ij}
15 for all i, j, k ∈ M do {
16  bad := {⟨x, x'⟩ | x ∈ g_ji ∧ x' ∈ g_ik ∧ ⟨x', t'⟩ ∉ g'_ik
17            ∧ M_i, {⟨x, t = 0⟩} ⊨ true U {⟨x', t'⟩}}
18  Guess B_1, B_2 s.t.  B_1 × B_2 ⊇ bad
19  g_ji := g_ji − B_1;  g_ik := g_ik − B_2
20 }
21 if (Verify(MDS, φ_S, ⟨(g_ij)_{i,j∈M}⟩))
22    return "synthesis successful"
23 else return "synthesis failed"
```

Figure 6: Procedure for solving the switching logic synthesis problem v2.

where $R(S)$ is the set of states obtained by resetting the t-component of every state in the set S to 0. If ϕ is a formula denoting the set S, then $R(\phi)$ is $\exists s(\phi[s/t] \wedge t = 0)$ (the notation $\phi[s/t]$ means replace t by s in ϕ). Informally, $R(\phi)$ can be computed by first removing facts about t from ϕ and then adding the new fact $t = 0$ to it.

The guards synthesized by Procedure SwitchSyn1 will use the new state variable t. However t was not part of our original problem specification. In the third step, the variable t is eliminated from the guards synthesized by Procedure SwitchSyn1. Suppose $S' := \langle (g'_{ij})_{i,j\in M}\rangle$ is the switching logic synthesized by Procedure SwitchSyn1. We first project out the t-component from S' to get our first guess for the desired S. Then, for every mode i, and for each entry guard, say g'_{ji}, and for each exit guard g'_{ik}, we compute pairs of states $(\mathbf{x}, \mathbf{x}')$ such that $\mathbf{x} \in g_{ji}$, $\mathbf{x}' \in g_{ik}$, there is a trajectory in mode i that starts from state $\langle \mathbf{x}, t = 0\rangle$ and reaches $\langle \mathbf{x}', t'\rangle$ in time t', and $\langle \mathbf{x}', t'\rangle$ is not in g'_{ik}. A behavior where mode i in entered in state $\mathbf{x}$ and exited in $\mathbf{x}'$ was disallowed in S', but it is allowed in S (since S ignores t). Hence, we need to either remove $\mathbf{x}$ from g_{ji}, or remove $\mathbf{x}'$ from g_{ik}. Procedure SwitchSyn2 procedure non-deterministically makes this choice.

Removal of states from the guards can potentially cause the modified switching logic to become unsafe. Hence, in the final step, we need to verify that the updated guards still satisfy Equation 6. This is performed by the function `Verify`. The function `Verify` can be implemented by calling Procedure SwitchSyn1 and checking its return value.

We can now state the soundness and completenss of Procedure SwitchSyn2 for solving the switching logic synthesis problem in Def. 4.

Theorem 3 (Soundness of Procedure SwitchSyn2). *If Procedure* SwitchSyn2 *terminates with "synthesis success-*

ful" and g'_{ij} are the discovered guards, then the hybrid system $\texttt{HS} := (\texttt{MDS}, \langle (g'_{ij})_{i,j \in M} \rangle)$ *is safe for* ϕ_S *and it satisfies the dwell time requirements specified by* `Te` *and* `Tx`.

PROOF. (Sketch) The final `Verify` check guarantees that `HS` is safe. By Theorem 1, the switching logic S' synthesized by Procedure SWITCHSYN1 on Line 8 satisfies the dwell time requirements. In the third phase, we explicitly ensure that the projected guards g_{ij}'s admit the same trajectories that guards g'_{ij}'s admitted. Hence, dwell time requirements continue to hold for the new guards. □

We can also state and prove completeness of Procedure SWITCHSYN2.

THEOREM 4 (COMPLETENESS OF PROCEDURE SWITCHSYN2). *If, for every possible guess on Line 18, the Procedure* SWITCHSYN2 *terminates with "synthesis failed", then there is no* $S' \subseteq S$ *such that the hybrid system* $\texttt{HS} := (\texttt{MDS}, S')$ *is safe and it satisfies the dwell time requirements.*

PROOF. (Sketch) The first phase just transforms the problem to an extended `MDS` and by Theorem 2, we know that we do not lose any solutions in the second phase (Line 8). Hence, the only place where completeness might be compromised in the third phase. However, we make non-deterministic guesses and hence we can always guess the correct solution, if one exists. This gives us the desired completeness result. □

Procedure SWITCHSYN2 is nondeterministic and involves making the correct guesses in the postprocessing stage. We can get a deterministic version of the procedure by making arbitrary guesses at each point. This deterministic version will be sound: whenever the procedure outputs "synthesis successful", the synthesis problem in Definition 4 indeed has a positive answer. However, it will not be complete: even when there is a positive answer for the synthesis problem, the deterministic variant can fail to find the appropriate guards because it can make the wrong choices. Some form of backtracking appears to be required. In practice, our implementation's heuristically-guided choices have always obtained a positive answer.

5. LEARNING GUARDS FROM SIMULATIONS

A key step in the implementation of Algorithms SWITCHSYN1 and SWITCHSYN2 is the computation of the *bad* state sets. In general, since the mode dynamics can be non-linear and quite complex, exactly computing the *bad* sets through analytical means is computationally infeasible. However, it is easier to perform numerical simulation of even complex, non-linear dynamics from individual points. In particular, in many cases, numerical simulation can be used to check whether a point $\mathbf{x}$ is a member of *bad*. Given such a membership check, our approach uses machine learning to compute an over-approximation of *bad*. While such over-approximation can result in a loss of completeness, it is guaranteed to generate safe switching logic.

5.1 Machine Learning

Our procedure assumes the availability of a machine learning algorithm $\mathcal{L}$ that can learn any target set from a *concept class* $\mathcal{C}$. $\mathcal{L}$ uses an oracle that can label points $\mathbf{x}$ as being in the target concept (i.e., $\mathbf{x} \in$ *bad*) or not in it (i.e. $\mathbf{x} \notin$ *bad*).

$\mathcal{L}$ is parameterized by $\mathcal{C}$, a point we sometimes make explicit by writing $\mathcal{L}_\mathcal{C}$ rather than $\mathcal{L}$.

Formally, given the following three inputs: (i) an over-approximation $\overline{c} \in \mathcal{C}$ of the set *bad*; (ii) a simulation oracle that can label a point $\mathbf{x}$ as $\mathbf{x} \in$ *bad* or $\mathbf{x} \notin$ *bad*; and (iii) (optionally) a sample of examples $P \subseteq$ *bad* (if they exist), $\mathcal{L}_\mathcal{C}$ must generate as output a set $out_\mathcal{L} \in \mathcal{C}$ with the following properties: if *bad* $\in \mathcal{C}$, then $out_\mathcal{L} =$ *bad*; otherwise, $out_\mathcal{L} \supseteq$ *bad*.

For simplicity, we describe below how $\mathcal{L}$ can be implemented when *bad* is an interval constraint on a single variable. It is possible to extend this method to conjunctions of interval constraints on multiple variables. An exploration of extensions to more complicated sets is left to future work.

5.2 Simulation Oracles

We assume the availability of the following two kinds of simulation-based oracles:

- *Oracle* $\mathcal{SO}_A$*:* This is an oracle that, given a state $\mathbf{x}$, the dynamics of a mode $Mode_i$, and state sets ϕ_1 and ϕ_2, returns a Boolean answer indicating whether the following property holds:

  $$Mode_i, \{\mathbf{x}\} \models (\phi_1 \mathbf{U} \phi_2)$$

  Note that definition of $\mathcal{SO}_A$ is motivated by the need to compute *bad* in Line 8 of Procedure SWITCHSYN1.
- *Oracle* $\mathcal{SO}_B$*:* This is an oracle that, given a state pair $\langle \mathbf{x}, \mathbf{x}' \rangle$, the dynamics of a mode $Mode_i$, extended-state set ψ, and state sets ϕ_1 and ϕ_2, returns a Boolean answer indicating whether the following property holds:

  $$\begin{aligned} & \mathbf{x} \in \phi_1 \wedge \mathbf{x}' \in \phi_2 \wedge \langle \mathbf{x}', t' \rangle \notin \psi \\ \wedge \quad & Mode_i, \{\langle \mathbf{x}, 0 \rangle\} \models (true \mathbf{U} \langle \mathbf{x}', t' \rangle) \end{aligned}$$

  The definition of $\mathcal{SO}_B$ is motivated by the need to compute *bad* in Line 17 of Procedure SWITCHSYN2.

Implementing these oracles involves performing a simulation from state $\mathbf{x}$ according to the (deterministic) dynamics in $Mode_i$, checking whether the condition on the RHS of the $\mathbf{U}$ operator has become true, and if not, checking that the LHS condition remains true. We assume the presence of a numerical simulator that can, for the mode dynamics of interest, select an appropriate discretization of time so as to check the above formulas with the $\mathbf{U}$ operator.

5.3 Learning Interval Constraints

We now describe how one can implement $\mathcal{L}$ for learning an interval constraint over a single variable $x \in X$. This form of constraint suffices for learning guards for all examples we consider in this paper. We give conditions under which the algorithm presented here satisfies the conditions required of $\mathcal{L}$ as stated above in Sec. 5.1.

An interval constraint is of the form $x \in [l_i, u_i]$ where $l_i, u_i \in \mathbb{Q}$. This constraint can also be expressed using inequalities as $l_i \leq x \leq u_i$.

Thus, $\mathcal{C}$ is the set of all constraints of the form $x \in [l_i, u_i]$ for any $l_i, u_i \in \mathbb{Q}$ and for any $x \in X$. The initial over-approximation $\overline{c}$ and the set $out_\mathcal{L}$ generated by $\mathcal{L}$ are both representable as an interval constraint.

Algorithm $\mathcal{L}_\mathcal{C}$ begins by checking the end-points of $\overline{c} = [\overline{l}, \overline{u}]$ for membership in *bad*. If both $\overline{l}$ and $\overline{u}$ are in *bad*, it simply outputs $out_\mathcal{L} = \overline{c}$. Otherwise, it selects the minimum and maximum elements $x_{\min}$ and $x_{\max}$ in the set of examples $P \in$ *bad*. (If P is not provided as input, $\mathcal{L}$ will randomly sample elements of $out_\mathcal{L}$ until an example $P \in$ *bad* is found).

$\mathcal{L}$ then performs binary search in the ranges $[\overline{l}, x_{\min}]$ and $[x_{\max}, \overline{u}]$ until it finds two examples $x_l \in [\overline{l}, x_{\min}]$ and $x_u \in [x_{\max}, \overline{u}]$ such that $x_l, x_u \in bad$ where x_l is the smallest such point and x_u is the largest. It then outputs $out_{\mathcal{L}} = [x_l, x_u]$.

It is easy to see that if $bad \in \mathcal{C}$, then $out_{\mathcal{L}} = bad$.

However, if $bad \notin \mathcal{C}$, then bad must be a disjoint union of intervals. Under the condition that P contains one point from each interval in this union, we obtain $out_{\mathcal{L}} \supset bad$.

Alternatively, suppose that the dynamics within each mode i is such that each state variable evolves monotonically with time – i.e., its value within that mode either increases with time or it decreases, but not both. In this case, bad cannot be a disjoint union of intervals, and so $out_{\mathcal{L}} = bad$. All examples discussed in this paper have this monotonicity property.

5.4 Discussion

We make some remarks on the above procedure.

First, note that restricting $out_{\mathcal{L}}$ to be an interval constraint does not require the final guards to also be of this form, since the designer is free to specify a *starting* switching logic using arbitrary expression syntax. The restriction only means that the set of points *removed* from the guards at each iteration of the fixpoint computation must be representable as an interval constraint to avoid losing completeness by removing too many points. As we demonstrate in our experimental results, we are able to synthesize interesting and non-trivial switching logic in spite of this restriction to the guard syntax.

Next, we observe that to employ the binary search procedure, we need to discretize the domains of variables in X. In general, such discretization is induced by a corresponding discretization of time chosen by the numerical simulator. Since controllers are in any case implemented using finite-precision computer arithmetic, we believe this finitization of intervals is not a restriction in practice.

Finally, we note that it is possible to extend the above procedure to learn a *conjunction of interval constraints*, viz., where $\mathcal{C}$ is the set of all n-dimensional boxes in $\mathbb{R}^n$ [7]. In the case that bad is not of this form, an over-approximation is obtained by applying the procedure in Sec. 5.3 to each $x \in X$ separately and taking the disjunction of the generated intervals.

6. EXPERIMENTS

We have implemented our technique using a Matlab-based numerical simulator. Apart from the *Thermostat Controller* described earlier in Section 3, we present two other case studies: *Traffic Collision and Avoidance System* and *Automated Transmission System*. The total runtime of synthesis for Thermostat Controller v1, v2 Case A, v2 Case B, TCAS Case A, Case B and Automated Transmission was 21.6, 26.2, 25.7, 55.3, 59.1 and 83.6 seconds respectively. More detailed experimental results are presented in the full version [7].

6.1 Traffic Collision and Avoidance System

Consider a simplified version of the Traffic Collision and Avoidance System (TCAS) [15], which seeks to ensure that two planes flying in opposite directions do not collide and maintain a specified safe distance (200 meters in our example). It operates by guiding the planes through a turn-left/fly-straight/turn-right maneuver as shown in the Figure 7. The three recovery maneuvers are indicated by corresponding mode names. We need to synthesize switching logic between the modes such that the planes are always atleast 200 meters apart at all times.

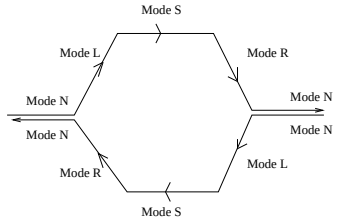

Figure 7: Simplified Traffic Collision and Avoidance System

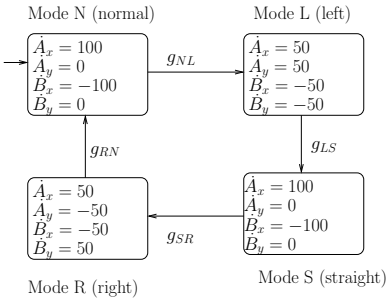

Figure 8: Simplified Traffic Collision and Avoidance System

The dynamics of the four modes of TCAS are given in Figure 7. We limit the movement of the plane in 2 dimensions $(X - Y)$ to simplify the example. Let $(\dot{A}_x, \dot{A}_y)$, and $(\dot{B}_x, \dot{B}_y)$ denote the (X, Y) velocities of the two planes A and B. Let $d(A, B)$ denote the Euclidean distance between the two planes, that is, $d(A, B) = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2}$. Hence we have the following safety property: $d(A, B) \geq 200$. In addition to this safety property, we also require that the planes at the end of the maneuver must regain their original orientation, that is, along the X-axis. So, $A_y = 0$ and $B_y = 0$ when returning to the normal mode at the end of the maneuver. Further, we would like to switch away from the straight mode only after the planes have crossed each other, that is, $A_x - B_x > 0$. We initialize the guards as given in Equation 8 using the safety property and the other specifications mentioned above.

$$\begin{aligned} g^0_{NL} &: d(A, B) \geq 200 \\ g^0_{LS} &: d(A, B) \geq 200 \\ g^0_{SR} &: d(A, B) \geq 200 \wedge A_x - B_x > 0 \\ g^0_{RN} &: d(A, B) \geq 200 \wedge A_y = 0 \wedge B_y = 0 \end{aligned} \tag{8}$$

Consider two cases for the synthesis problem - one with just the minimum dwell-time constraint and the second with both the minimum and the maximum dwell-time constraint. This example illustrates how designers can use maximum dwell-time constraints to synthesize systems with desired behavior and not just safe behavior.

Case A: Only a minimum dwell-time requirement of 1 second in the straight mode is provided, ensuring that the planes spend some time in the straight mode before turning again. The final guards synthesized by computing fixpoint

are as follows.

$$\begin{aligned} g_{NL} &: g^0_{NL} \wedge B_x - A_x \geq 283 \\ g_{LS} &: g^0_{LS} \wedge A_y - B_y \geq 200 \\ g_{SR} &: g^0_{SR} \wedge A_x - B_x \geq 117 \\ g_{RN} &: g^0_{RN} \wedge (A_x - B_x \geq 0 \vee B_x - A_x \geq 283) \end{aligned} \quad (9)$$

The behavior of the system synthesized above is illustrated in Figure 9. The initial state is $A_x = 0, A_y = 0, B_x = 600, B_y = 0$. X and Y denote the distance between the planes in X and Y co-ordinates and D denotes the distance between the planes. The minimum value of D is 200m. The synthesized system is safe and satisfies the minimum dwell-time requirement but it has the undesirable behavior of switching from normal mode to maneuver modes immediately at the initial state. The planes could have delayed their entry into the maneuver mode.

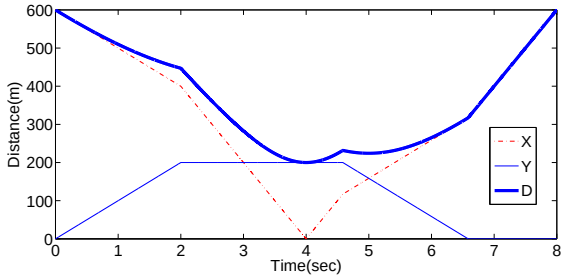

Figure 9: Sample Behavior of Synthesized TCAS

Case B: In this case, we also provide a maximum dwell-time requirement of 1.1 second in the straight mode. This ensures that the planes fly towards each other till it is necessary to switch to maneuver modes. By specifying the maximum dwell-time requirement on the straight mode, we effectively limit the time spend in maneuver and hence, force the system to stay in the normal mode for a longer time. The final guards synthesized by computing the fixpoint are as follows.

$$\begin{aligned} g_{NL} &: g^0_{NL} \wedge 303 \geq B_x - A_x \geq 283 \\ g_{LS} &: g^0_{LS} \wedge A_y - B_y \geq 200 \wedge B_x - A_x \leq 103 \\ g_{SR} &: g^0_{SR} \wedge A_x - B_x \geq 117 \\ g_{RN} &: g^0_{RN} \wedge (A_x - B_x \geq 0 \vee B_x - A_x \geq 283) \end{aligned} \quad (10)$$

We again plot the behavior of the synthesized system with the same initial state as Case A in Figure 10. The time spent in maneuver is now limited and we stay in normal mode till the planes are 303 meters far from each other and then switch to the collision avoidance maneuver.

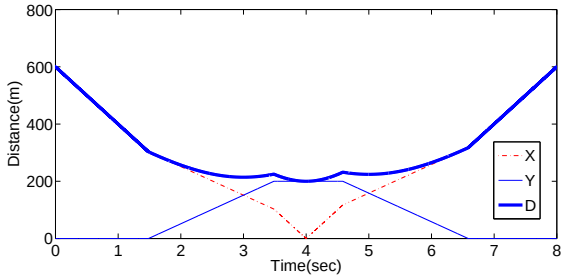

Figure 10: Sample Behavior of Synthesized TCAS with Max Dwell-time

6.2 Automatic Transmission

Our final example is a 3-gear *automated transmission* system [10]. The transmission system is illustrated in Figure 11; notice that *the mode dynamics are non-linear*. u and d denote the throttle in accelerating and deaccelerating mode. The transmission efficiency η is η_i when the system is in mode i.

$$\eta_i = 0.99e^{-(\omega - a_i)^2/64} + 0.01$$

where $a_1 = 10, a_2 = 20, a_3 = 30$ and ω is the speed. The distance covered is denoted by θ. The acceleration in mode i is given by the product of the throttle and transmission efficiency. For simplicity, we fix $u = 1$ and $d = -1$. From an initial state of $\theta = 0, \omega = 0$, the system must reach $\theta = \theta_{max} = 1700$ with $\omega = 0$. The synthesis problem is to find the guards between the modes such that the efficiency η is high for speeds greater than some threshold, that is, $\omega \geq 5 \Rightarrow \eta \geq 0.5$. Also, ω must be less than an upper limit of 60. So, the safety property ϕ_S to be enforced would be

$$(\omega \geq 5 \Rightarrow \eta \geq 0.5) \wedge (0 \leq \omega \leq 60)$$

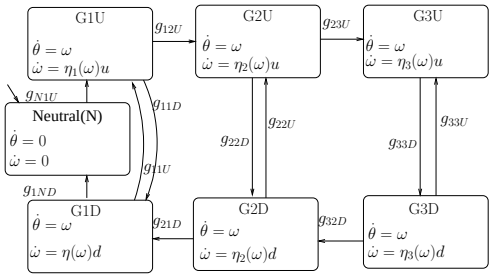

Figure 11: Automatic Transmission System

Since the speed must reduce to 0 on reaching θ_{max}, the guard g_{1ND} is initialized to $\phi_S \wedge \theta = \theta_{max} \wedge \omega = 0$. All the other guards are initialized to ϕ_S. The final set of guards obtained after fixpoint computation are as follows.

$$\begin{aligned} g_{N1U}, g_{11U} &: 0 \leq \omega \leq 16.70 \\ g_{12U}, g_{22U} &: 13.29 \leq \omega \leq 26.70 \\ g_{23U}, g_{33U} &: 23.29 \leq \omega \leq 36.70 \ , \ g_{33D} : 23.29 \leq \omega \leq 36.70 \\ g_{32D}, g_{22D} &: 13.29 \leq \omega \leq 26.70 \\ g_{21D}, g_{11D} &: 0 \leq \omega \leq 16.70 \ , \ ; g_{1ND} : \theta = \theta_{max} \wedge \omega = 0 \end{aligned} \quad (11)$$

We now impose a minimum dwell-time of 5 seconds on all the six gear modes. The guards obtained by computing the fixpoint are as follows.

$$\begin{aligned} g_{N1U} &: \omega = 0 \ , \ g_{11U} : \omega = 0 \\ g_{1ND} &: \theta = \theta_{max} \wedge \omega = 0 \ , \ g_{12U} : 13.29 \leq \omega \leq 23.42 \\ g_{11D} &: 1.31 \leq \omega \leq 16.70 \ , \ g_{23U} : 26.70 \leq \omega \leq 33.42 \\ g_{22D} &: \omega = 26.70 \ , \ g_{33D} : \omega = 36.70 \\ g_{32D} &: 16.58 \leq \omega \leq 26.70 \ , \ g_{33U} : 23.29 \leq \omega \leq 33.42 \\ g_{21D} &: 1.31 \leq \omega \leq 16.70 \ , \ g_{22U} : 13.29 \leq \omega = 23.42 \end{aligned} \quad (12)$$

The plot of the behavior of the transmission system when it is made to switch from Neutral mode through the six gear modes and back to the Neutral mode is shown in Figure 12. The efficiency η is always greater than 0.5 when the speed is higher than 5 and we spend atleast 5 seconds in the six gear modes. Starting from $\theta = 0, \omega = 0$, the synthesized system reaches $\theta = \theta_{max}$ with $\omega = 0$.

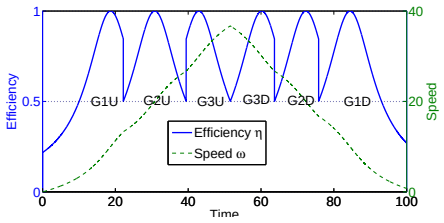

Figure 12: Transmission efficiency and speed with changing gears

7. RELATED WORK

Past work on synthesis of switching logic can be broadly classified into two categories depending on the goals of synthesis. The first category finds controllers that meet some liveness specifications, such as synthesizing a trajectory to drive a hybrid system from an initial state to a desired final state [9, 11]. The second category finds controllers that meet some safety specification [2]. Our work combines both safety specifications with min-dwell requirements (which is a form of liveness specification) to enable synthesis of systems that meet some performance related properties.

Past techniques for synthesis of switching logic involve computing the set of controlled reachable states either in the style of solving a game [2, 19] or some abstraction based reasoning [14, 3, 17]. They all perform some kind of iterative fixpoint computation and are limited in the kind of continuous dynamics they can handle. The novelty of our work lies in presenting a new technique based on combining local simulation inside a mode with fix-point computation across modes. Our simulation-based approach to reason about the continuous dynamics inside each mode makes our approach more generally applicable. Simulations have been used to perform verification [8, 5, 4], but we use simulations to perform synthesis. Recently, [18] proposed a constraint-based technique for synthesizing switching logic that involves generating and solving an $\exists\forall$ constraint (as opposed to performing a fixpoint computation). However, the size of the constraint increases as the number of modes increase. In our approach, reasoning is performed on one mode at a time and hence it scales better than [18].

Dwell time is a well-known concept in hybrid systems [6, 12, 13], where it has been used for verification. We use dwell time for synthesis. The user can use it to guarantee synthesis of nonzeno and desirable systems.

Our problem formulation has the high-level philosophy of "completing a partially-specified design" also explored in other domains, such as software synthesis by sketching [16]. To our knowledge, however, the approach we take, combining verification, learning, and simulation, is distinct and novel.

8. CONCLUSION

We presented a new approach for synthesizing safe hybrid systems that uses numerical simulations and fixpoint computation. The user can guide synthesis by specifying dwell time requirements and the form of the guards. Extension of the approach to synthesize optimal designs and with richer guards is left for future work.

Acknowledgments

The UC Berkeley authors were supported in part by NSF grants CNS-0644436 and CNS-0627734, by an Alfred P. Sloan Research Fellowship, and by the Multiscale Systems Center (MuSyC), one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program. The fourth author was supported in part by NSF grants CNS-0720721 and CSR-0917398 and NASA grant NNX08AB95A.

9. REFERENCES

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, February 1995.

[2] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli. Effective synthesis of switching controllers for linear systems. In *Proceedings of the IEEE*, volume 88, pages 1011–1025, 2000.

[3] J. Cury, B. Krogh, and T. Niinomi. Synthesis of supervisory controllers for hybrid systems based on approximating automata. In *IEEE Transactions on Automatic Control*, pages 564–568, 1998.

[4] A. Donze and O. Maler. Systematic simulation using sensitivity analysis. In *HSCC*, volume 4416 of *LNCS*, pages 174–189, 2007.

[5] A. Girard and G. J. Pappas. Verification by simulation. In *HSCC*, volume 3927 of *LNCS*, pages 272–286, 2006.

[6] T. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Trans. on Automatic Control*, 43:540–554, 1998.

[7] S. Jha, S. Gulwani, S. A. Seshia, and A. Tiwari. Synthesizing switching logic for safety and dwell-time requirements. Technical Report UCB/EECS-2010-28, EECS Department, University of California, Berkeley, March 2010.

[8] J. Kapinski, B. H. Krogh, O. Maler, and O. Stursberg. On systematic simulation of open continuous systems. In *HSCC*, volume 2623 of *LNCS*. Springer, 2003.

[9] T. J. Koo, G. J. Pappas, and S. Sastry. Mode switching synthesis for reachability specifications. In *HSCC*, pages 333–346, 2001.

[10] J. Lygeros. Lecture notes on hybrid systems. 2004.

[11] P. Manon and C. Valentin-Roubinet. Controller synthesis for hybrid systems with linear vector fields. In *IEEE International Symposium on Intelligent Control*, pages 17–22, 1999.

[12] S. Mitra, D. Liberzon, and N. Lynch. Verifying average dwell time of hybrid systems. *ACM Trans. Embedded Comput. Syst.*, 8(1), 2008.

[13] C. Mitrohin, A. Podelski, and S. Wagner. Dwell time refinement, 2009. Personal communication.

[14] T. Moor and J. Raisch. Discrete control of switched linear systems. In *European Control Conference*, 1999.

[15] G. J. Pappas, C. Tomlin, and S. Sastry. Conflict resolution for multi-agent hybrid systems. In *IEEE Control and Decision Conference*, pages 1184–1189, 1996.

[16] A. Solar-Lezama, L. Tancau, R. Bodik, S. A. Seshia, and V. A. Saraswat. Combinatorial sketching for finite programs. In *ASPLOS*, pages 404–415, 2006.

[17] P. Tabuada. Controller synthesis for bisimulation equivalence. *Systems and Control Letters*, 57(6):443–452, 2008.

[18] A. Taly, S. Gulwani, and A. Tiwari. Synthesizing switching logic using constraint solving. In *VMCAI*, pages 305–319, 2009.

[19] C. J. Tomlin, J. Lygeros, and S. S. Sastry. A game theoretic approach to controller design for hybrid systems. In *Proceedings of the IEEE*, volume 88, pages 949–970, 2000.