pdfauthor

Verifying Average Dwell Time of Hybrid Systems

Sayan Mitra
Massachusetts Institute of Technology
Daniel Liberzon
University of Illinois at Urbana-Champaign
and
Nancy Lynch
Massachusetts Institute of Technology

The switched system model abstracts away the discrete mechanisms of a hybrid system in terms of an exogenous switching signal. *Dwell Time* and *Average Dwell Time (ADT)* criteria, introduced by Morse and Hespanha, define restricted classes of switching signals that guarantee stability of the whole system, provided the individual modes of the switched system are stable. In this paper, we present a set of techniques for establishing stability through verification of ADT properties. We introduce a new type of simulation relation for hybrid automata—*switching simulation*—that allows us to show that the ADT of one automaton is no less than that of another. We show that the question of whether a given hybrid automaton has ADT τ_a can be answered by checking a carefully designed invariant or by solving an optimization problem. The invariant-based method is applicable to any hybrid automaton. For suitable classes of automata the invariant in question can be checked automatically. The optimization-based method is applicable to a restricted class of initialized hybrid automata. For this class, a solution of the optimization problem either gives a counterexample execution that violates the ADT property, or it confirms that the automaton indeed satisfies the property. The optimization-based approach is automatic and complements the invariant-based method in the sense that they can be used in combination to find the unknown ADT of a given hybrid automaton.

Categories and Subject Descriptors: D. Software [**D.2 Software Engineering**]: D.2.4. Software/Program Verification

General Terms: Hybrid Systems, Stability, Verification

Additional Key Words and Phrases: Hybrid systems, Simulation relation, Optimization-based verification

1. INTRODUCTION

Rapid growth in communication and microprocessor technologies is fueling the development of complex embedded devices which are being deployed to perform

Author's addresses: S. Mitra and N. Lynch, Computer Science and Artificial Intelligence Laboratory, Massachusetts Inst. of Technology, 32 Vassar Street, Cambridge, MA 02139, USA. Email: {mitras,lynch}@csail.mit.edu D. Liberzon, Coordinated Science Laboratory, Univ. of Illinois at Urbana-Champaign, Urbana, IL 61801, U.S.A. Email: liberzon@uiuc.edu
This work is supported by the MURI project:DARPA/AFOSR MURI F49620-02-1-0325 grant.

increasingly more critical and sophisticated tasks. Stability of such devices and systems is often a natural requirement. For digital control systems, stability is of fundamental importance and has been an area of intense research. Stability properties are also important in distributed computation and control type applications. For instance, a set of mobile robots starting from arbitrary locations in the plane are required to coordinate and converge to some formation; a set of failure prone processes communicating over unreliable channels are expected to perform some useful computation once the failures cease and the message delays become normal. Both the above requirements can be phrased as stability properties of the respective systems.

The standard approach for describing complex embedded systems consisting of software components and physical processes is to assume that the state space of the system is partitioned into finite number of equivalence classes. Each equivalence class is called a *mode*. We denote the set of modes by $\mathcal{P}$. The evolution of the state $\mathbf{x}$ in mode i, for some $i \in \mathcal{P}$, is described by some differential equation of the form $d(\mathbf{x}) = f_i(\mathbf{x})$. Mode transitions are described by guards and and reset maps. Hybrid automata-like models (see e.g. [Alur et al. 1995; Lynch et al. 2003]) embody the above point of view. Verification of safety properties of hybrid automata through reachable set computations and deductive techniques have received a lot of attention in the recent years (see, e.g., [Mitchell and Tomlin 2000; Prajna and Jadbabaie 2004; Kurzhanski and Varaiya 2000; Henzinger and Majumdar 2000; Livadas et al. 1999; Heitmeyer and Lynch 1994; Mitra et al. 2003] .

Analyzing the stability of hybrid automata is challenging because the stability of the continuous dynamics of each individual mode does not necessarily imply the stability of the whole automaton. The basic tool for studying stability relies on the existence of a *Common Lyapunov function*, whose derivative along the trajectories of all the modes must satisfy suitable inequalities. When such a function is not known or does not exist, *Multiple Lyapunov* functions [Branicky 1998] are useful for proving stability of a chosen execution. These and many other stability results are based on the *switched system* [Liberzon 2003; van der Schaft and Schumacher 2000] view of hybrid systems.

Switched systems may be seen as higher-level abstractions of hybrid automata. A switched system model neglects the details of the discrete mechanisms of hybrid automata, namely the guards and the reset maps, and instead relies on an exogenous switching signal to bring about the mode switches. Assuming that the switching signal belongs to a certain class, one can focus on the stability of a hybrid system with respect to its continuous dynamics. If the individual modes of the automaton are stable, then the *Dwell Time* [Morse 1996] and the more general *Average Dwell Time (ADT)* criteria of [Hespanha and Morse 1999] define restricted classes of switching signals that guarantee stability of the whole system.

To be more specific, a hybrid automaton $\mathcal{A}$ has ADT τ_a if, in every execution fragment of $\mathcal{A}$, any τ_a interval of time, on an average, has at most one mode switch. A large average dwell time means that the system spends enough time in each mode, so as to dissipate the transient energy gained through mode switches. Having a large average dwell time itself is not sufficient for stability; in addition, the individual modes of the automaton must also be stable. In fact, the problem of proving the

stability of a hybrid system can be broken down into, (a) finding Lyapunov functions for the individual modes, and (b) checking the appropriate ADT property. In this paper we assume that a solution to part (a)—a set of Lyapunov functions for the individual modes—is known from existing techniques from systems theory [Khalil 2002], and we present semi-automatic methods for proving the ADT properties. Thus, we provide a missing piece in the toolbox for analysis of stability of hybrid systems.

1.1 Contributions

In order to verify ADT, we have to use a model that captures both the discrete and continuous mechanisms of hybrid systems. We use the *Structured Hybrid Automaton (SHA)* model derived from the *Hybrid Input/Output Automaton (HIOA)* framework of [Lynch et al. 2003]. First of all, we define what it means for a given SHA to switch "faster than" another SHA with respect to ADT. We introduce a new kind of simulation relation, called *switching simulation*, which gives a sufficient condition for establishing the above "faster than" relationship between pairs of automata. That is, if automaton $\mathcal{A}_1$ is simulated by a faster automaton $\mathcal{A}_2$, and $\mathcal{A}_2$ has ADT τ_a, we can conclude that the ADT of $\mathcal{A}_1$ is at least τ_a. This gives us a sound method for abstraction that is particularly geared towards ADT verification.

Our first method for ADT verification relies on checking invariant properties. In order to check if automaton $\mathcal{A}$ has ADT τ_a, we transform it to a new automaton $\mathcal{A}(\tau_a)$, such that $\mathcal{A}$ has ADT τ_a if and only if $\mathcal{A}(\tau_a)$ has a particular invariant property $\mathcal{I}(\tau_a)$. We can then appeal to suitable invariant checking tools, like HyTech [Henzinger et al. 1997], PHAVer [Frehse 2005b] or PVS [Owre et al. 1996], to check $\mathcal{I}(\tau_a)$. This method is applicable to general SHAs; however, the invariant $\mathcal{I}(\tau_a)$ can be checked automatically only for restricted classes of automata. For hybrid automata that are not amenable to automatic invariant checking, semi-automatic deductive techniques can be used.

Our second method for ADT verification is based on solving optimization problems. To check if automaton $\mathcal{A}$ has ADT τ_a, we formulate an optimization problem $\mathsf{OPT}(\tau_a)$. From the solution of $\mathsf{OPT}(\tau_a)$ we either get a counterexample execution fragment of $\mathcal{A}$ that violates the ADT property τ_a, or else we get a proof that no such counterexample exists, and that $\mathcal{A}$ has ADT τ_a. We show that for certain classes of SHAs $\mathsf{OPT}(\tau_a)$ can indeed be formulated and solved using standard mathematical programming techniques. The optimization-based method complements the invariant method because the two can be combined to find the ADT of SHAs.

1.2 Organization

The rest of the paper is organized as follows: in Section 2 the Structured Hybrid Automaton (SHA) model is formally introduced and briefly compared to other existing models; the linguistic conventions used throughout the paper for describing SHAs are presented; stability and Average Dwell Time (ADT) properties of SHAs are defined. The section concludes with the definition of switching simulation relations and the soundness theorem. Section 3 presents the invariant-based approach for verifying ADT. The necessary transformations are presented and the ADT verification of two hybrid systems—a scale-independent hysteresis switch and a leaking gas burner—are described. Section 4 presents the optimization-based method for

verifying ADT. At first, the optimization problem OPT corresponding to ADT verification is stated. It is established that for initialized rectangular SHAs and for more restricted one-clock initialized SHAs OPT can be solved effectively. These results along with switching simulation relations are used to automatically verify ADT of a linear hysteresis switch and a thermostat with nondeterministic switches. For initialized rectangular SHAs, a Mixed Integer Linear Program formulation for solving OPT is presented. Section 5 concludes this paper with a summary of contributions and with some remarks interesting directions for future research.

2. STRUCTURED HYBRID AUTOMATA, STABILITY AND AVERAGE DWELL TIME

This section forms the mathematical basis for the rest of the paper. First we introduce the *Structured Hybrid Automata (SHA)* model, a simplified version of the HIOA model of [Lynch et al. 2003] tailored for the results in this paper. In Section 2.3 we introduce the linguistic conventions used throughout the paper to describe SHAs. In Section 2.4 we define the different notions of stability and the role of Average Dwell Time criterion in stability analysis. Finally, in Section 2.6 we define *switching simulation relations* for SHAs and show that they provide a sufficient condition for proving equivalence of SHAs with respect to ADT.

Several models for hybrid systems have been proposed in the literature. For instance, the *Hybrid Automaton* model of [Alur et al. 1995] is well established; the switched system model [Liberzon 2003] has been widely used to obtain many stability related results; the *General Hybrid Dynamical System* of [Branicky 1995; Branicky et al. 1998] is proposed with particular emphasis of controller design. We will not dwell on the relationship of the SHA model with all the above, however, we briefly note the features of the SHA model that make it suitable for this paper.

First, the SHA model imposes a variable structure on the state-space. Indeed, this adds some notational overhead, but it is a convenient feature for modelling hybrid systems whose discrete state consists of not just locations (or modes) but interesting data structures such as, counters, queues and heaps. Secondly, between the Hybrid Automaton model of [Alur et al. 1995] and SHA, the latter is closer to the switched system model because it provides direct handle on the trajectories and it does not require built in structures (such as guards and reset maps) for describing the discrete mechanism. Thus, SHA is more suitable for adopting results from the theory of switched systems such as stability via ADT. Further, owing to the structure of SHAs invariants and simulation relations can be proved inductively by a case analysis on the actions and the trajectories. Even for systems where fully automatic verification is impossible, such proofs can be partially automated using theorem provers [Mitra and Archer 2005]. Finally, the HIOA framework, of which SHA is a part, provides powerful compositionality theorems. We do not make use of composition in this paper, however, in the future when we study external stability and input to state stability, we can do so within the same mathematical framework.

2.1 Variables and trajectories

We denote the domain of a function f by $f.dom$. For a set $S \subseteq f.dom$, we write $f \lceil S$ for the restriction of f to S. If f is a function whose range is a set of functions and Y is a set, then we write $f \downarrow Y$ for the function g with $g.dom = f.dom$ such

that for each $c \in g.dom$, $g(c) = f(c) \lceil Y$. For a tuple or an array b with n elements, we refer to its i^{th} element by $b[i]$.

We fix the *time axis* T to be $\mathbb{R}_{\geq 0}$. For any $J \subseteq \mathsf{T}$ we define $J+t = \{t'+t \mid t' \in J\}$. In the SHA model a variable structure is used to specify states. Let X be a set of state variables; X is partitioned into X_d, the set of *discrete variables*, and X_c, the set of *continuous variables*. Each variable $x \in X$ is associated with a *type*, which is the set of values that x can assume. Each $x \in X_d$ (respectively, X_c) has *dynamic type*, which is the pasting closure of the set of constant (resp. continuous) functions[1] from left-closed intervals in T to the type of x. A *valuation* $\mathbf{x}$ for the set of variables X is a function that associates each $x \in X$ to a value in its type. The set of all valuations of X is denoted by $val(X)$. A *trajectory* $\tau : J \to val(X)$ specifies the values of all variables in X over a time interval J with left endpoint of J equal to 0, with the constraint that evolution of each $x \in X$ over the trajectory should be consistent with its dynamic type. The set of all trajectories for the set of variables X is denoted by $traj(X)$.

A trajectory with domain $[0, 0]$ is called a *point trajectory*. The *limit time* of a trajectory τ, written as $\tau.ltime$, is the supremum of $\tau.dom$. If $\tau.dom$ is right closed then τ is *closed*. The *first state* of τ, $\tau.fstate$ is $\tau(0)$, and if τ is closed, then the *last state* of τ, $\tau.lstate$, is $\tau(\tau.ltime)$.

Given a trajectory τ and $t \in \mathsf{T}$, the function $(\tau + t) : (\tau.dom + t) \to X$ is defined as $(\tau + t)(t') := \tau(t' - t)$, for each $t' \in (\tau.dom + t)$. Given two trajectories τ_1 and τ_2, τ_1 is a *prefix* of τ_2, written as $\tau_1 \leq \tau_2$, if $\tau_1 = \tau_2 \lceil \tau_1.dom$. Also, τ_1 is a *suffix* of τ_2 if $\tau_1 = (\tau_2 \lceil [t, \infty)) - t$, for some $t \in \tau_2.dom$. If τ_1 is a closed trajectory with $\tau_1.ltime = t$ and $\tau_2.fstate = \tau_1.lstate$, then the function $\tau_1 \frown \tau_2 : \tau_1.dom \cup (\tau_2.dom + t) \to X$ is defined as $\tau_1(t)$ if $t \leq u$ and $\tau_2(t - u)$ otherwise.

A set of trajectories $\mathcal{T}$ for X is *closed under prefix (suffix)* if for any $\tau \in \mathcal{T}$ a prefix (suffix) τ' of τ is also in $\mathcal{T}$. Suppose $\tau \in traj(X)$ and let x be some variable name in X. With some abuse of notation we define the function $x : \tau.dom \to type(x)$ to be $x(t) := (\tau \downarrow x)(t)$, for every $t \in \tau.dom$.

2.2 Structured hybrid automata

The Structured Hybrid Automaton model is derived from the Hybrid Input/Output Automaton (HIOA) model of [Lynch et al. 2003]. We are concerned with internal stability of hybrid systems in this paper, so SHAs do not have input/output variables and do not distinguish among input, output, and internal actions. On the other hand, this model describes the trajectories of automata using "state models" that are collections of differential and algebraic equations, instead of abstract sets of functions.

Definition 2.1. A *state model* F for a set of variables X is a set of differential equations for X_c of the form $d(\mathbf{x}_c) = f(\mathbf{x}_c)$, such that: (1) For every $\mathbf{x} \in val(X)$, there exists solution τ of F with $\tau.fstate = \mathbf{x} \lceil X_c$, and (2) for all $t \in \tau.dom$, $(\tau \downarrow X_d)(t) = (\tau \downarrow X_d)(0)$. The prefix and suffix closure of the set of trajectories of X that satisfy the above conditions is denoted by $traj(X, F)$.

[1] This set of functions must be closed under time-shift, restriction to subintervals, and pasting. See [Kaynar et al. 2005] for formal definition of these closure properties

Definition 2.2. A *Structured Hybrid Automaton* (SHA) is a tuple $\mathcal{A} = (X, Q, \Theta, A, \mathcal{D}, P)$, where (1) X is a set of *variables*, including a special discrete variable called *mode*. (2) $Q \subseteq val(X)$ is the set of states, (3) $\Theta \subseteq Q$ is a nonempty set of *start states*, (4) A is a set of *actions*, (5) $\mathcal{D} \subseteq Q \times A \times Q$ is a set of *discrete transitions*, and (6) P is an indexed family F_i, $i \in \mathcal{P}$, of state models, where $\mathcal{P}$ is an index set.

A transition $(\mathbf{x}, a, \mathbf{x}') \in \mathcal{D}$ is written in short as $\mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}'$ or as $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ when $\mathcal{A}$ is clear from context. A transition $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ is called a *mode switch* if $\mathbf{x} \lceil mode \neq \mathbf{x}' \lceil mode$. The set of *mode switching transitions* is denoted by $M_{\mathcal{A}}$. The *precondition* of action a is $Pre_a := \{\mathbf{x} \in Q \mid \exists \mathbf{x}', \mathbf{x} \xrightarrow{a} \mathbf{x}' \in \mathcal{D}\}$.

In this paper, we assume that the right hand sides of the differential equations in the state models are well behaved (locally Lipschitz), and the differential equations have solutions defined globally in time. Therefore, for each F_i, $i \in \mathcal{P}$ and $\mathbf{x} \in Q$ with $\mathbf{x} \lceil mode = i$, there exists a trajectory τ starting from $\mathbf{x}$ that satisfies F_i and if $\tau.dom$ is finite then $\tau.lstate \in Pre_a$ for some $a \in A$. The set $\mathcal{T}$ of trajectories of SHA $\mathcal{A}$ is defined as $\mathcal{T} := \bigcup_{i \in \mathcal{P}} traj(X, F_i)$.

An *execution fragment* captures a particular run of $\mathcal{A}$; it is defined as an alternating sequence of actions and trajectories $\alpha = \tau_0 a_1 \tau_1 a_2 \ldots$, where (1) each $\tau_i \in \mathcal{T}$, and (2) if τ_i is not the last trajectory then $\tau_i.lstate \xrightarrow{a_{i+1}} \tau_{i+1}.fstate$. The *first state* of an execution fragment α, $\alpha.fstate$, is $\tau_0.fstate$. An execution fragment α is an *execution* of $\mathcal{A}$ if $\alpha.fstate \in \Theta$. The *length* of a finite execution fragment α is the number of actions in α. An execution fragment is *closed* if it is a finite sequence, and the domain of the last trajectory is closed. Given a closed execution fragment $\alpha = \tau_0, a_1, \ldots, \tau_n$, its *last state*, $\alpha.lstate$, is $\tau_n.lstate$ and its *limit time*, $\alpha.ltime$, is defined as $\sum_{i=0}^{n} \tau_i.ltime$. A closed execution fragment α of SHA $\mathcal{A}$ is a *cycle* if $\alpha.fstate = \alpha.lstate$. We define the following shorthand notation for the valuation of the variables of $\mathcal{A}$ at $t \in [0, \alpha.ltime]$, $\alpha(t) := \alpha'.lstate$, where α' is the longest prefix of α with $\alpha'.ltime = t$.

A state $\mathbf{x} \in Q$ is *reachable* if it is the last state of some execution of $\mathcal{A}$. An execution fragment α is *reachable* if $\alpha.fstate$ is reachable. An *invariant property* or simply an *invariant* of $\mathcal{A}$ is a condition on X that holds in all reachable states of $\mathcal{A}$. An invariant property $\mathcal{I}$ can be proved inductively by showing: (a) for all $\mathbf{x} \in \Theta$, $\mathcal{I}(\mathbf{x})$, (b) for all $\mathbf{x} \xrightarrow{a} \mathbf{x}'$, if $\mathcal{I}(\mathbf{x})$ then $\mathcal{I}(\mathbf{x}')$, and (c) for all closed trajectories $\tau \in \mathcal{T}_{\mathcal{A}}$, if $\mathcal{I}(\tau.fstate)$, then $\mathcal{I}(\tau.lstate)$.

2.3 Linguistic conventions

The standard circle-arrow diagrams used for specifying hybrid automata become a little cumbersome for automata with many modes and discrete transitions. In this paper, we use an extension of the TIOA Language [Kaynar et al. 2005] to specify SHAs. We briefly describe the semantics of this language using the code in Figure 1 as an example.

Variable names, their static and dynamic types, and initial values are defined in the **Variables** section (lines 1–3). All but the real valued variables are considered to be discrete; a real valued discrete variable is declared using the **Discrete** keyword.

Action names are declared in the **Actions** section (lines 5–6) and the corresponding transitions are defined in the **Transitions** section using the precondition-effect style. The predicate following the **Precondition** keyword after action a defines

Pre_a. The assignment statements after the keyword **Effect** define the relation between pre and the post state of the corresponding transition.

The **Trajectories** section (lines 21–29) defines the state model, the invariants and the stopping condition for each mode. For example, in the leaking mode, the continuous variables evolve according to the simple differential equation $d(x) = 1$ (line 27); the *stopping condition* $x = D_2$ following the **Stop when** keyword, means that if $(\tau \downarrow x)(t) = D_2$, for some $t \in \tau.dom$, then t is the limit time of τ.

Burner(D_1, D_2), where $D_1, D_2 \in \mathbb{R}_{\geq 0}$

Variables:
2 $mode \in \{normal, leaking\}$, **initially** $normal$
$x, z \in R$, **initially** $x = 0$
4
Actions
6 leak, repair

8 **Transitions**:
leak
10 **Precondition**
$mode = normal \wedge x \geq D_1$
12 **Effect**
$mode \leftarrow leaking$, $x \leftarrow 0$

13
repair
Precondition 15
$mode = leaking \wedge x = D_2$
Effect 17
$mode \leftarrow leaking$, $x \leftarrow 0$
19

Trajectories: 21
Trajdef normal
Evolve $d(x) = 1$ 23

Trajdef leaking 25
Invariant $x \leq D_2$
Evolve $d(x) = 1$ 27
Stop when $x = D_2$

Fig. 1: Leaking gas burner

2.4 Stability and ADT

We adopt the standard stability definitions [Khalil 2002] and state them in the language of SHAs. Stability is a property of the continuous variables of SHA $\mathcal{A}$, with respect to the standard Euclidean norm in R^n which we denote as $|\cdot|$. We assume that each state model $F_i \in P$ of $\mathcal{A}$ has the origin as its common equilibrium point, that is, $F_p(0) = 0$ for all $i \in \mathcal{P}$. The origin is a *stable* equilibrium point of a SHA $\mathcal{A}$, in the sense of Lyapunov, if for every $\epsilon > 0$, there exists a $\delta > 0$, such that for every execution α of $\mathcal{A}$,

$$|\alpha(0)| \leq \delta \Rightarrow |\alpha(t)| \leq \epsilon \quad \forall t \quad 0 \leq t \leq \alpha.ltime, \tag{1}$$

and we say that $\mathcal{A}$ is *stable*. An SHA $\mathcal{A}$ is *asymptotically stable* if it is stable and δ can be chosen so that

$$|\alpha(0)| \leq \delta \Rightarrow \alpha(t) \to 0 \quad as \quad t \to \infty \tag{2}$$

If the above condition holds for all δ then $\mathcal{A}$ is *globally asymptotically stable*.

Uniform stability is a concept which guarantees that the stability property in question holds not just for executions, but for any execution fragment. $\mathcal{A}$ is uniformly stable in the sense of Lyapunov, if for every $\epsilon > 0$ there exists a constant $\delta > 0$, such that for any execution fragment α,

$$|\alpha(t_0)| \leq \delta \Rightarrow |\alpha(t)| \leq \epsilon, \forall t_0, t, \ 0 \leq t_0 \leq t \leq \alpha.ltime$$

An SHA $\mathcal{A}$ is said to be uniformly asymptotically stable if it is uniformly stable and there exists a $\delta > 0$, such that for every $\epsilon > 0$ there exists a T, such that for any execution fragment α,

$$|\alpha(t_0)| \leq \delta \Rightarrow |\alpha(t)| \leq \epsilon, \quad \forall t \geq t_0 + T \tag{3}$$

It is said to be *globally uniformly asymptotically stable* if the above holds for all δ, with $T = T(\delta, \epsilon)$. All the above stability properties are by definition uniform over executions.

It is well known that a switched system is stable if all the individual subsystems are stable and the switching is sufficiently slow, so as to allow the dissipation of the transient effects after each switch. The *dwell time* [Morse 1996] and the *average dwell time* [Hespanha and Morse 1999] criteria define restricted classes of switching signals, based on switching speeds, and one can conclude the stability of a system with respect to these restricted classes.

Definition 2.3. Given a duration of time $\tau_a > 0$, SHA $\mathcal{A}$ has *Average Dwell Time (ADT)* τ_a if there exists a positive constant N_0, such that for every reachable execution fragment α,

$$N(\alpha) \leq N_0 + \alpha.ltime/\tau_a, \tag{4}$$

where $N(\alpha)$ is the number of mode switches in α. The number of *extra switches* of α with respect to τ_a is defined as $S_{\tau_a}(\alpha) := N(\alpha) - \alpha.ltime/\tau_a$.

Theorem 1 from [Hespanha and Morse 1999], adapted to SHA, gives a sufficient condition for stability based on average dwell time. Roughly, it states that a hybrid system is stable if the discrete switches are between modes which are individually stable, provided that the switches do not occur *too frequently on the average*. See Section 3.2 of [Liberzon 2003] for a proof[2].

Theorem 2.4. *Suppose there exist positive definite, radially unbounded, and continuously differentiable functions $\mathcal{V}_i : \mathbb{R}^n \to \mathbb{R}^n$, for each $i \in \mathcal{P}$, and positive numbers λ_0 and μ such that:*

$$\frac{\partial \mathcal{V}_i}{\partial \mathbf{x}_c} f_i(\mathbf{x}_c) \leq -\lambda_0 \mathcal{V}_i(\mathbf{x}_c), \quad \forall \mathbf{x}_c, \quad \forall i \in \mathcal{P}, \text{and}$$

$$\mathcal{V}_i(\mathbf{x}'_c) \leq \mu \mathcal{V}_j(\mathbf{x}_c), \quad \forall \mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}', \text{ where } i = \mathbf{x}' \lceil \text{mode and } j = \mathbf{x} \lceil \text{mode}.$$

Then, $\mathcal{A}$ is globally uniformly asymptotically stable if it has an ADT $\tau_a > \frac{\log \mu}{\lambda_0}$.

This stability condition effectively allows us to decouple the construction of Lyapunov functions—the $\mathcal{V}_i$'s for each $i \in \mathcal{P}$, which we assume are known from available methods of system theory—from the problem of checking that every execution of the automaton has a certain average dwell time.

2.5 ADT verification overview

In general, it is hard to prove properties like the ADT property which are quantified over all the executions of an automaton. Our first approach for verifying ADT

[2]In [Hespanha and Morse 1999] and [Liberzon 2003] this theorem is presented for the case when discrete transitions do not change the valuation of the continuous variables, but the same proof establishes the result stated here.

relies on simple transformations defined by a small set of history variables, that are sufficient to convert the ADT property to an equivalent invariant. This enables us to prove ADT properties using the techniques available for proving invariants. Our second method is based on a complementary approach. We attempt to find an execution of the automaton, that violates (4). We show that this search can be formulated as an optimization problem, and for certain restricted classes of SHA, we show that the optimization problem can be solved efficiently.

The invariant-based approach is generally applicable to any SHA. Automated invariant checking tools are only available for restricted classes of SHAs. For cases where automated invariant checking is not possible, we can inductively prove the invariants. Such proofs can be partially automated using theorem provers (see e.g., [Mitra and Archer 2005]). On the other hand, the optimization-based approach is applicable for classes of SHAs where the resulting optimization problem can be solved using available mathematical programming techniques. When applicable, this approach yields an automatic method for verifying ADT.

The two methods for verifying ADT can be combined to find ADT of a given SHA as follows: we can start with some candidate value of $\tau_a > 0$ and search for a counterexample execution fragment for it using the optimization-based approach. If such an execution fragment is found, then we decrease τ_a (say, by a factor of 2) and try again. If eventually the optimization approach fails to find a counterexample execution fragment for a particular value of τ_a, then we use the invariant approach to try to prove that this value of τ_a is an ADT for the given system.

2.6 Equivalence with respect to ADT: Switching simulations

To check if τ_a is an ADT for a given SHA $\mathcal{A}$, it is often easier to check the same ADT property for another, more abstract, SHA $\mathcal{B}$ that is "equivalent" to $\mathcal{A}$ with respect to switching behavior. Formally, given SHAs $\mathcal{A}$ and $\mathcal{B}$, if for all $\tau_a > 0$, τ_a is an ADT for $\mathcal{B}$ implies that τ_a is an ADT for $\mathcal{A}$, then we write this as $\mathcal{A} \geq_{ADT} \mathcal{B}$. If $\mathcal{B} \geq_{ADT} \mathcal{A}$ and $\mathcal{A} \geq_{ADT} \mathcal{B}$ then we say $\mathcal{A}$ and $\mathcal{B}$ are *equivalent with respect to switching*.

In this section we will develop a simulation relation-based method for proving the above equivalence relationship. Traditionally, simulation relations have been widely used to prove that the set of visible behavior of one automaton is included in that of another automaton [Lynch and Vaandrager 1996]. Simulation relations have been used for verifying safety and timing-based properties of hybrid systems (see e.g., [Frehse 2005a; Lynch 1996; Weinberg and Lynch 1996; Lim et al. 2005]). For ADT verification, the "visible" part of an execution we are concerned with is the number of mode switches that occur and the amount of time that elapses over the execution. Following this intuition we define a new kind of simulation relation that gives us a way of inductively proving the $\geq_{ADT}$ relationship between a pair of SHAs.

Definition 2.5. Consider SHAs $\mathcal{A}$ and $\mathcal{B}$. A *switching simulation relation* from $\mathcal{A}$ to $\mathcal{B}$ is a relation $\mathcal{R} \subseteq Q_{\mathcal{A}} \times Q_{\mathcal{B}}$ satisfying the following conditions, for all states $\mathbf{x}$ and $\mathbf{y}$ of $\mathcal{A}$ and $\mathcal{B}$, respectively:

(1) If $\mathbf{x} \in \Theta_{\mathcal{A}}$ then there exists a state $\mathbf{y} \in \Theta_{\mathcal{B}}$ such that $\mathbf{x}\ \mathcal{R}\ \mathbf{y}$.

(2) If $\mathbf{x}\ \mathcal{R}\ \mathbf{y}$ and α is an execution fragment of $\mathcal{A}$ with $\alpha.fstate = \mathbf{x}$ and consisting

of one single action surrounded by two point trajectories, then $\mathcal{B}$ has a closed execution fragment β, such that $\beta.fstate = \mathbf{y}$, $N(\beta) \geq 1$, $\beta.ltime = 0$, and $\alpha.lstate \; \mathcal{R} \; \beta.lstate$.

(3) If $\mathbf{x} \; \mathcal{R} \; \mathbf{y}$ and α is an execution fragment of $\mathcal{A}$ with $\alpha.fstate = \mathbf{x}$ and consisting of a single closed trajectory, then $\mathcal{B}$ has a closed execution fragment β, such that $\beta.fstate = \mathbf{y}$, $\beta.ltime \leq \alpha.ltime$, and $\alpha.lstate \; \mathcal{R} \; \beta.lstate$.

Lemma 2.6. *If $\mathcal{A}$ and $\mathcal{B}$ be SHAs, and let $\mathcal{R}$ be a switching simulation relation from $\mathcal{A}$ to $\mathcal{B}$, then for all $\tau_a > 0$ and for every execution α of $\mathcal{A}$, there exists an execution β of $\mathcal{B}$ such that $S_{\tau_a}(\beta) \geq S_{\tau_a}(\alpha)$.*

PROOF. We fix τ_a and α and construct an execution of $\mathcal{B}$ that has more extra switches than α. Let $\alpha = \tau_0 a_1 \tau_1 a_2 \tau_2 \ldots$ and let $\alpha.fstate = \mathbf{x}$. We consider cases:

(1) α is an infinite sequence.
We can write α as an infinite concatenation $\alpha_0 \frown \alpha_1 \frown \alpha_2 \ldots$, in which the execution fragments α_i with i even consist of a trajectory only, and the execution fragments α_i with i odd consist of a single discrete transition surrounded by two point trajectories.
We define inductively a sequence $\beta_0 \beta_1 \beta_2 \ldots$ of closed execution fragments of $\mathcal{B}$ such that $\mathbf{x} \; \mathcal{R} \; \beta_0.fstate$, $\beta_0.fstate \in \Theta_{\mathcal{B}}$, and for all i, $\beta_i.lstate = \beta_{i+1}.fstate$, $\alpha_i.lstate \; \mathcal{R} \; \beta_i.lstate$, and $S_{\tau_a}(\beta) \geq S_{\tau_a}(\alpha)$. Property 1 of the definition of switching forward simulation ensures that there exists such a $\beta_0.fstate$ because $\alpha_0.fstate \in \Theta_{\mathcal{A}}$. We use Property 3 of the definition of switching simulation for the construction of the β_i's with i even. This gives us $\beta_i.ltime \leq \alpha_i.ltime$ for every even i. We use Property 2 of the definition of switching simulation for the construction of the β_i's with i odd. This gives us $\beta_i.ltime = \alpha_i.ltime$ and $N(\beta_i) \geq N(\alpha_i)$ for every odd i. Let $\beta = \beta_0 \frown \beta_1 \frown \beta_2 \ldots$. Since $\beta_0.fstate \in \Theta_{\mathcal{B}}$, β is an execution for $\mathcal{B}$. Since $\beta.ltime \leq \alpha.ltime$ and $N(\beta) \geq N(\alpha)$, the required property follows.

(2) α is a finite sequence ending with a closed trajectory.
Similar to first case.

(3) α is a finite sequence ending with an open trajectory.
The final open trajectory of β is constructed using a concatenation of infinitely many smaller and smaller closed trajectories. The proof of this case is also similar to first case, except that the open trajectory of β is constructed using Lemma 4.22 of [Kaynar et al. 2005].

□

Theorem 2.7. *If $\mathcal{A}$ and $\mathcal{B}$ are SHAs and $\mathcal{R}$ is a switching simulation relation from $\mathcal{A}$ to $\mathcal{B}$, then $\mathcal{B} \geq_{ADT} \mathcal{A}$.*

PROOF. We fix a τ_a. Given N_0 such that for every execution β of $\mathcal{B}$, $S_{\tau_a}(\beta) \leq N_0$, it suffices to show that for every execution α of $\mathcal{A}$, $S_{\tau_a}(\alpha) \leq N_0$. We fix α. From Lemma 2.6 we know that there exists a β such that $S_{\tau_a}(\beta) \geq S_{\tau_a}(\alpha)$, from which the result follows. □

Corollary 2.8. *Let $\mathcal{A}$ and $\mathcal{B}$ be SHAs, and let $\mathcal{R}$ be a switching forward simulation relation both from $\mathcal{A}$ to $\mathcal{B}$ and from $\mathcal{B}$ to $\mathcal{A}$. Then, $\mathcal{A}$ and $\mathcal{B}$ are equivalent with respect to switching.*

3. VERIFYING ADT: INVARIANT APPROACH

In this section we present a method for verifying ADT of SHAs which relies on checking invariants. In Sections 3.2 and 3.3 we use this method to verify ADT of a simple leaking gas-burner and a scale-independent hysteresis switch.

3.1 Transformations for ADT verification

To prove that a given SHA $\mathcal{A} = (X, Q, \Theta, A, \mathcal{D}, P)$ has average dwell time τ_a, we transform it to a new SHA $\mathcal{A}(\tau_a) = (X_1, Q_1, \Theta_1, A_1, \mathcal{D}_1, P_1)$ as follows:

(1) $X_1 = X \cup \{q, y\}$, where $q \in \mathbb{Z}$ is a counter and $y \in \mathbb{R}_{\geq 0}$ is a timer.
(2) $\Theta_1 = \{(\mathbf{x}, q, y) \mid \mathbf{x} \in \Theta, q = 0, y = 0\}$
(3) $A_1 = A \cup \{\mathsf{decrement}.\}$.
(4) $\mathcal{D}_1$ has the following transitions:

$$
\begin{aligned}
&i. \quad \forall\, \mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}' \in \mathcal{D} \setminus M_{\mathcal{A}}, q \in \mathbb{Z}, y \in \mathbb{R}_{\geq 0} \quad (\mathbf{x}, q, y) \xrightarrow{a}_{\mathcal{A}(\tau_a)} (\mathbf{x}', q, y),\\
&ii. \quad \forall\, \mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}' \in M_{\mathcal{A}}, q \in \mathbb{Z}, y \in \mathbb{R}_{\geq 0}, (\mathbf{x}, q, y) \xrightarrow{a}_{\mathcal{A}(\tau_a)} (\mathbf{x}', q+1, y),\\
&iii. \quad \forall\, \mathbf{x} \in Q, q \in \mathbb{Z}, \quad (\mathbf{x}, q, \tau_a) \xrightarrow{\mathsf{decrement}}_{\mathcal{A}(\tau_a)} (\mathbf{x}, q-1, 0).
\end{aligned}
$$

(5) P_1 is obtained by adding to each state model in P the differential equation $d(y) = 1$ and the stopping condition $y = \tau_a$.

Informally, the counter q increments every time there is a mode switch of $\mathcal{A}$, and the timer reduces the count by 1 in every τ_a time by triggering the $\mathsf{decrement}$ action. For every trajectory $\tau \in \mathcal{T}'$, the restriction of τ on the set of variables X is a trajectory of $\mathcal{A}$, and $d(y) = 1$, and if $(\tau \downarrow y)(t) = \tau_a$ then $\tau.ltime = t$.

Lemma 3.1. *If τ_a is not an ADT for automaton $\mathcal{A}$, then for every $N_0 \in \mathbb{N}$ there exists a closed execution α of $\mathcal{A}$, such that $N(\alpha) > N_0 + \alpha.ltime/\tau_a$.*

Proof. Let us fix N_0. Automaton $\mathcal{A}$ does not have ADT τ_a, so we know that there exists an execution α of $\mathcal{A}$ such that $N(\alpha) > N_0 + \alpha.ltime/\tau_a$. If α is infinite, then there is a closed prefix of α that violates (4). If α is finite and open, then the closed prefix of α excluding the last trajectory of α violates (4). □

Theorem 3.2. *Given $\tau_a > 0$, all executions of $\mathcal{A}$ have ADT τ_a if and only if there exists $N_0 \in \mathbb{N}$ such that $q \leq N_0$ is an invariant for $\mathcal{A}(\tau_a)$.*

Proof. From Lemma 3.1 we know that it is sufficient to show that all closed executions of $\mathcal{A}$ satisfy (4) if and only if $q \leq N_0$ is an invariant for $\mathcal{A}(\tau_a)$. For the "if" part, consider a closed execution α of $\mathcal{A}$ and let α' be the "corresponding" execution of $\mathcal{A}(\tau_a)$. Let $\mathbf{x}' = \alpha.lstate$, from the invariant we know that $\mathbf{x}' \lceil q \leq N_0$. From construction of $\mathcal{A}(\tau_a)$ we know that $N(\alpha) = N(\alpha')$ and $\alpha'.ltime = \alpha.ltime$ and therefore $\mathbf{x}' \lceil q = N(\alpha') - \lfloor \frac{\alpha'.ltime}{\tau_a} \rfloor$. It follows that $N(\alpha) - \frac{\alpha.ltime}{\tau_a} \leq N_0$.

For the "only if" part, consider a reachable state $\mathbf{x}'$ of $\mathcal{A}(\tau_a)$. There exists a closed execution α' such that $\mathbf{x}'$ is the last state of α'. Let α be an execution of $\mathcal{A}$ "corresponding" to α'. Since $N(\alpha) \leq N_0 + \lfloor \frac{\alpha.time}{\tau_a} \rfloor$ implies $N(\alpha') \leq N_0 + \lfloor \frac{\alpha'.ltime}{\tau_a} \rfloor$, it follows that $\mathbf{x}' \lceil q \leq N_0$. □

Remark. In Equation (4), the number N_0 can be arbitrary. Thus to show that a given τ_a is an average dwell time of an automaton, we need to show that q is bounded uniformly over all executions.

The above transformation is acceptable for asymptotic stability, but it allows q to become negative and then rapidly return to zero. So, it does not guarantee uniform stability. For uniform stability we want all reachable execution fragments of $\mathcal{A}$ to satisfy (4). Consider any reachable execution fragment α of $\mathcal{A}$, with $\alpha.ftime = t_1$, and $\alpha.ltime = t_2$. Let $N_{\tau_a}(t_2, t_1)$ and $S_{\tau_a}(t_2, t_1)$ denote the number of switches and the number of extra switches over α with respect to average dwell time τ_a. Thus, every reachable execution fragment α of $\mathcal{A}$ satisfies (4), if

$$N_{\tau_a}(t_2, t_1) \leq N_0 + \frac{t_2 - t_1}{\tau_a}, \text{ or } \quad S_{\tau_a}(t_2, t_1) \leq N_0,$$

where $t_1 = \alpha.ftime$, and $t_2 = \alpha.ltime$. We introduce an additional variable q_{min} which stores the magnitude of the smallest value ever attained by q. For uniform stability we need to show that the total change in q between any two reachable states is bounded by N_0. Instead of introducing the new variable q_{min} we could restrict the variable q to have only non-negative values, to obtain uniform stability[3].

Theorem 3.3. *Given $\tau_a > 0$, all reachable execution fragments of $\mathcal{A}$ have ADT τ_a if and only if $q - q_{min} \leq N_0$ is an invariant for $\mathcal{A}(\tau_a)$.*

Proof. From Lemma 3.1 we know that it suffices to consider closed execution fragments only. For the "if" part, consider a reachable closed execution fragment α of $\mathcal{A}$ which is a part of the execution β, such that $\alpha.fstate = \beta(t_1)$ and $\alpha.lstate = \beta(t_2)$. Let α' and β' be the "corresponding" execution (fragment) of $\mathcal{A}(\tau_a)$. Based on whether or not q_{min} changes over the interval $[t_1, t_2]$, we have the following two cases:

If q_{min} does not change in the interval, then $\beta'(t_1) \lceil q_{min} = \beta'(t_2) \lceil q_{min} = \beta'(t) \lceil q$ for some $t_{min} < t_1$, and $q(t_2, t_1) = q(t_2, t_{min}) - q(t_1, t_{min}) \leq q(t_2, t_{min})$. Since $\beta'(t_2)$ satisfies the invariant, $q(t_2, t_{min}) = \beta'(t_2) \lceil q - \beta'(t_2) \lceil q_{min} \leq N_0$ from which we get $q(t_2, t_1) \leq N_0$.

Otherwise, there exists some $t_{min} \in [t_1, t_2]$, such that $\beta'(t_2) \lceil q_{min} = \beta'(t_{min}) \lceil q < \beta'(t_1) \lceil q_{min}$, and $q(t_2, t_1) = q(t_2, t_{min}) + q(t_{min}, t_1) \leq q(t_2, t_{min})$. Again, from the invariant property at $\beta'(t_2)$, we get $q(t_2, t_1) \leq q(t_2, t_{min}) \leq N_0$.

For the "only if" part, let $\mathbf{x}'$ be a reachable state, and ξ' be a closed execution of $\mathcal{A}(\tau_a)$, such that $\mathbf{x}' = \xi'.lstate$. Further, let ξ be the "corresponding" execution of $\mathcal{A}$, and t_0 be the intermediate point where q attains its minimal value over ξ, that is, $\xi(t) \lceil q_{min} = \xi(t_0) \lceil q$. Since ξ is a reachable execution fragment of $\mathcal{A}$, it satisfies Equation (4), and we have: $N(t, t_0) \leq N_0 + \frac{t - t_0}{\tau_a}$. Rewriting,

$$q(t, 0) + \frac{t}{\tau_a} - q(t_0, 0) - \frac{t_0}{\tau_a} \leq N_0 + \frac{t - t_0}{\tau_a}$$

By assumption, $q(t_0, 0) = \xi'(t) \lceil q_{min} = \mathbf{x}' \lceil q_{min}$, therefore, it follows that $\mathbf{x}' \lceil q - \mathbf{x}' \lceil q_{min} \leq N_0$. □

3.2 Leaking gas burner

The above transformations yield a simple method for verifying ADT properties of SHA. We illustrate this with the toy leaking gas-burner system from [Alur et al.

[3]We thank Andy Teel for suggesting this alternative formulation.

1993]. The specification of the leaking gas-burner automaton is given in Figure 1 in Section 2.3. The gas-burner system has two modes, the normal mode and the leaking mode, and it switches between these two modes according to the following two rules. Every leak continues for D_2 seconds after which it is repaired and the system returns to the normal mode, and no leak occurs within the next D_1 seconds; it is known that $D_2 < D_1$. Mode switches are brought about by the leak and repair actions. Indeed, the ADT of this simple hybrid automaton is $\frac{D_1+D_2}{2}$.

To check whether a given τ_a is an average dwell time for **Burner** we transform this automaton according to the transformation described in Section 3.1. As the dynamics of the continuous variables in this system are suitable for model checking, we use the HyTech tool [Henzinger et al. 1997] to check if the $q \leq N_0$ is an invariant property of the transformed automaton.

For $D_1 = 20, D_2 = 4, N_0 = 1000$ and for different values of τ_a, we check if $q \leq N_0$ is an invariant for the transformed **Burner**. HyTech tells us that $q \leq N_0$ is indeed an invariant for $\tau_a \leq 12$, and not otherwise. It follows that the ADT of **Burner** with the above parameters is 12.

3.3 Scale-independent hysteresis switch

We verify the ADT property of a more interesting hybrid system, namely a Scale-independent hysteresis switch. This switching logic unit is a subsystem of an adaptive supervisory control system taken from [Hespanha et al. 2003] (also Chapter 6 of [Liberzon 2003]). Our goal is to prove the ADT property of this switching logic, which guarantees stability of the overall supervisory control system. The above references also present a proof of this property by a different approach.

Let $\mathcal{P} = \{1, \ldots, m\}$, $m \in \mathbb{N}$, be the index set for for a family of controllers. An adaptive supervisory controller consists of a family of candidate controllers $u_i, i \in \{1, \ldots, m\}$, which correspond to the parametric uncertainty range of the plant in a suitable way. Such a controller structure is particularly useful when the parametric uncertainty is so large that robust control design tools are not applicable. The controller operates in conjunction with a set of on-line estimators that provide *monitoring signals* $\mu_i, i \in \{1, \ldots, m\}$. Intuitively, smallness of μ_i indicates high likelihood that i is the actual parameter value. Based on these signals, the switching logic unit changes the variable *mode*, which in turn determines the controller to be applied to the plant.

In building the SHA model **HSwitch** (see Figure 2), we consider the monitoring signals to be generated by differential equations of the form, $d(\mu) = f_i(\mu)$, where $i \in \{1, \ldots, m\}$, and μ is the vector of monitoring signals. Our analysis does not depend on the exact nature of these differential equations. Instead, we require the monitoring signals for each μ_i to be continuous, monotonically nondecreasing, satisfying the following lower and upper bounds:

$$\mu_i(0) \geq C_0 \tag{5}$$

$$\mu_{i^*}(t) \leq C_1 + C_2 e^{\lambda t}, \text{for some } i^* \in \{1, \ldots, m\} \tag{6}$$

where λ, C_0, C_1 and C_2 are positive constants. The switching logic unit implements scale-independent hysteresis switching as follows: at an instant of time when controller i is operating, that is, $mode = i$ for some $i \in \{1, \ldots, m\}$, if there exists a

$j \in \{1, \ldots, m\}$ such that $\mu_j(1+h) \leq \mu_i$, then the switching logic sets $mode = j$ and applies output of controller j to the plant. Here h is a fixed positive hysteresis constant.

Since the monitoring signals are specified in terms of nonlinear bounds and there are arbitrary number of modes, we cannot apply model checking techniques directly to this system. Instead, we inductively prove a sequence of invariants which together with Theorem 3.2 establish the ADT of the hysteresis switch to be at least $\frac{\log(1+h)}{m\lambda}$.

For the ease of this analysis we introduce several extra history variables to **HSwitch** in addition to those described at the beginning of Section 3.1. The result is the automaton **TRHSwitch** of Figure 3. Lines 4–6, lines 26–29 and line 38 correspond to the transformation of Section 3.1. Note that the timer y is not reset to 0 after every decrement and therefore it records the total time elapsed. The following additional variables are introduced for ease of analysis: c counts the number of mode switches; c_i counts the number of switches to mode i; μ_i^r stores the value of μ_i at the instant when $mode$ became equal to i for the r^{th} time. Initially $\mu_i^0 = \mu_i$, for all $i \in \{1, \ldots, m\}$, $\mu_{i_0}^1 = \mu_{i_0}$, where i_0 is the initial mode; the rest of the μ_i^rs are set to a null value $\perp$.

HSwitch(m, h) where $m \in \mathbb{N}, h \in \mathbb{R}_{\geq 0}$

Variables:
- $mode \in \{1, \ldots, m\}$, **initially** $mode = i_0$
- $\mu \in \mathbb{R}^m$, **initially** $\mu_i \geq C_0 \ \forall \ i \in \{1, \ldots, m\}$
- **Derived**
  - $\mu_{min} = \min_i\{\mu_i\}$

Actions:
- switch(i,j), $i,j \in \{1, \ldots, m\}$

Transitions:
- switch(i,j)
  - **Precondition**
    - $mode = i \wedge (1+h)\mu_j \leq \mu_i$
  - **Effect** $mode \leftarrow j$

Trajectories:
- **Trajdef** mode$_i$, $i \in \{1, \ldots, m\}$
- **Evolve** $d(\mu) = f_i(\mu)$
- **Stop when**
  - $\exists\ j \in \{1, \ldots, m\}$
  - **such that** $(1+h)\mu_j \leq \mu_i$

Fig. 2: Hysteresis switch

For simplicity of presentation, we prove the invariants required for asymptotic stability, and not uniform asymptotic stability. Accordingly the average dwell time property we get is over executions and not over execution fragments of the automaton. The first two invariants state some straightforward properties of the state variables.

Invariant 3.4. $q \leq c - \frac{y}{\tau_a} + 1$.

Invariant 3.5. *For all* $i, j \in \{1, \ldots, m\}$, *if* $mode = j$ *then* $\mu_j \leq (1+h)\mu_i$, *in addition if* $c_j > 0$ *and* $y_j = 0$ *then* $\mu_j \leq \mu_i$.

Invariant 3.6. *For all* $i \in \{1, \ldots, m\}$, $c_i \geq 2 \Rightarrow \mu_i^{c_i} \geq (1+h)\mu_i^{c_i - 1}$.

Proof. We fix some i in $\{1, \ldots, m\}$. The base case holds vacuously. For the induction step, since the invariant involves only discrete variables, we only have to consider discrete transitions of the form $\mathbf{x} \xrightarrow{a} \mathbf{x}'$, where $a = switch_i$. Let $\mathbf{x}$ ⌈

TRHSwitch(m, h, τ_a) where $m \in \mathbb{N}, h, \tau_a \in \mathbb{R}_{\geq 0}$

Variables:
- $mode \in \{1, \ldots, m\}$, **initially** $mode = i_0$
- $\mu \in \mathbb{R}^m$, **initially** $\mu_i \geq C_0 \ \forall \ i \in \{1, \ldots, m\}$
- $q \in \mathbb{Z}$, **initially** $q = 0$
- $k \in \mathbb{N}$, **initially** $k = 0$
- $y \in \mathbb{R}$, **initially** $y = 0$
- **Discrete** $\mu^k \in \mathbb{R} \cup \{\perp\}$, $k \in \mathbb{N}$,
  - **initially** $\mu^0 = \mu, \forall\ k \neq 0, \mu^k = \perp$
- $c, c_i \in \mathbb{N}$, $i \in \{1, \ldots, m\}$
  - **initially** $c = 0$, $c_{i_0} = 1, \forall\ i \neq i_0$, $c_i = 0$,

Derived variables
- $\mu_{min} = \min_i\{\mu_i\}$

Actions:
- switch(i,j), $i,j \in \{1, \ldots, m\}$
- decrement

Transitions:

switch(i,j)
Precondition
- $mode = i \wedge (1+h)\mu_j \leq \mu_i$

Effect $mode \leftarrow j$
- $c \leftarrow c + 1; c_j \leftarrow c_j + 1;$
- $q \leftarrow q + 1; \mu^{c_j} = \mu_j$

decrement
Precondition
- $y = (k + 1)\tau_a$

Effect
- $q \leftarrow q - 1; k \leftarrow k + 1$

Trajectories:

Trajdef mode$_i$, $i \in \{1, \ldots, m\}$
Evolve $d(\mu) = f_i(\mu)$
Stop when
- $\exists\ j \in \{1, \ldots, m\}$
- **such that** $(1 + h)\mu_j \leq \mu_i$
- $\vee\, y = (k + 1)\tau_a$

Fig. 3: Transformed hysteresis switch

$mode = j$ and $\mathbf{x}' \lceil c_i = r + 1$. That is, action a is the $(r + 1)^{st}$ switch to mode i. From the transition relation, we know that $(1 + h)(\mathbf{x} \lceil \mu_i) = \mathbf{x} \lceil \mu_j$. It follows that:

$$\mathbf{x}' \lceil \mu_i^{r+1} = \mathbf{x}' \lceil \mu_i = (1 + h)(\mathbf{x}' \lceil \mu_j) \tag{7}$$

Let $\mathbf{x}''$ be the post state of the r^{th} $switch_i$ action. From the first part of Invariant 3.5, $(1 + h)(\mathbf{x}'' \lceil \mu_j) \geq \mathbf{x}'' \lceil \mu_i = \mathbf{x}'' \lceil \mu_i^r$. From monotonicity of μ_i, $\mathbf{x}' \lceil \mu_i \geq \mathbf{x}'' \lceil \mu_i$. Since $\mathbf{x}'' \lceil \mu_i^r = \mathbf{x}' \lceil \mu_i^r$ (no $switch_i$ action in between), we get $\mathbf{x}' \lceil \mu_j \geq \mathbf{x}' \lceil \mu_i^r$. Combining this last inequality with (7) we get, $\mathbf{x}' \lceil \mu_i^{r+1} \geq (1 + h)(\mathbf{x}' \lceil \mu_i^r)$. □

Now we are ready to prove the main invariant property, which states that for a particular choice of τ_a, the value of the variable q is bounded by some constant.

Theorem 3.7. *Let* $\tau_a = \frac{\log(1+h)}{\lambda m}$. $q \leq N_0$ *is an invariant of* **TRHSwitch**, *where* $N_0 = 2 + m + \frac{m}{\log(1+h)} \log\left(\frac{C_1+C_2}{C_0}\right)$.

Proof. Consider any reachable state $\mathbf{x}$. We observe that, the counter c is incremented every time a $switch_i$ action occurs for any $i \in \{1, \ldots, m\}$, and for each $i \in \{1, \ldots, m\}$ the counter c_i is incremented when the corresponding $switch_i$ action occurs. If $\mathbf{x} \lceil c$ is less than m then the result follows immediately from Invariant 3.4. Otherwise, $\mathbf{x} \lceil c \geq m$ and there must be some $j \in \{1, \ldots, m\}$, such that mode j is visited more than $\lceil \frac{\mathbf{x}\lceil c-1}{m} \rceil$ times, that is, $\mathbf{x} \lceil c_j \geq \lceil \frac{\mathbf{x}\lceil c-1}{m} \rceil$. Therefore, from Invariant 3.6 we know that there exists j in $\{1, \ldots, m\}$ such that

$\mathbf{x} \lceil \mu_j^{c_j} \geq (1+h)^{\lceil \frac{\mathbf{x}\lceil c-1}{m} \rceil - 1}(\mathbf{x} \lceil \mu_j^1)$. Taking logarithm and rearranging we have,

$$\mathbf{x} \lceil c \leq 1 + m + \frac{m}{\log(1+h)} \log\left(\frac{\mathbf{x} \lceil \mu_j^{c_j}}{\mathbf{x} \lceil \mu_j^1}\right)$$

Let $\mathbf{x}'$ be the post state of the c_j^{th} $switch_j$ action, then $\mathbf{x} \lceil \mu_j^{c_j} = \mathbf{x}' \lceil \mu_j$. From the second part of Invariant 3.5 and monotonicity of the monitoring signals, it follows that, for all $k \in \{1, \ldots, m\}$, $\mathbf{x} \lceil \mu_j^{c_j} = \mathbf{x}' \lceil \mu_j^{c_j} \leq \mathbf{x}' \lceil \mu_k \leq \mathbf{x} \lceil \mu_k$. It follows that, for all $k \in \{1, \ldots, m\}$,

$$\mathbf{x} \lceil c \leq 1 + m + \frac{m}{\log(1+h)} \log\left(\frac{\mathbf{x} \lceil \mu_k}{\mathbf{x} \lceil \mu_j^1}\right).$$

Form monotonicity and property (5) of the monitoring signals, $\mu_j^1 \geq \mu_j^0 \geq C_0$. Therefore, for all $k \in \{1, \ldots, m\}$,

$$\begin{aligned}
\mathbf{x} \lceil c &\leq 1 + m + \frac{m}{\log(1+h)} \log\left(\frac{\mathbf{x} \lceil \mu_k}{C_0}\right). \\
&\leq 1 + m + \frac{m}{\log(1+h)} \log\left(\frac{C_1 + C_2 e^{\lambda\ (\mathbf{x} \lceil y)}}{C_0}\right), \quad \text{replacing } k \text{ with } i^* \text{ of (6)} \\
&\leq 1 + m + \frac{m}{\log(1+h)} \log\left(\frac{C_1 + C_2}{C_0}\right) + \frac{\lambda m\ (\mathbf{x} \lceil y)}{\log(1+h)}
\end{aligned}$$

Using Invariant 3.4, and putting $\tau_a = \frac{\log(1+h)}{\lambda m}$, we get the result. □

Remark. From the above invariant and Theorem 3.2 it is established that **HSwitch** has an average dwell time of at least $\frac{\log(1+h)}{\lambda m}$. To ensure stability of the overall supervisory control system, the parameters h and λ must be such that this average dwell time satisfies the inequality of Theorem 2.4. For details we refer the reader to Chapter 6 of [Liberzon 2003].

4. OPTIMIZATION BASED APPROACH

In this section we develop the second method for verifying ADT properties. From Definition 2.3 it follows that $\tau_a > 0$ is *not* an ADT of a given SHA $\mathcal{A}$ if and only if, for every $N_0 > 0$ there exists a reachable execution fragment α of $\mathcal{A}$ such that $S_{\tau_a}(\alpha) > N_0$. Thus, if we solve the following optimization problem:

$$\mathsf{OPT}(\tau_a): \qquad \alpha^* \in \arg\max S_{\tau_a}(\alpha)$$

over all the execution of $\mathcal{A}$, and the optimal value $S_{\tau_a}(\alpha^*)$ turns out to be bounded, then we can conclude that $\mathcal{A}$ has ADT τ_a. Otherwise, if $S_{\tau_a}(\alpha^*)$ is unbounded then we can conclude that τ_a is not an ADT for $\mathcal{A}$. However, $\mathsf{OPT}(\tau_a)$ may not be directly solvable because, among other things, the executions of $\mathcal{A}$ may not have finite descriptions. In the remainder of this paper we study particular classes of SHA for which $\mathsf{OPT}(\tau_a)$ can be formulated and solved efficiently.

4.1 One-clock initialized SHA

We consider a special class of SHA, called *one-clock initialized SHA*, for which $\mathsf{OPT}(\tau_a)$ can be solved using classical graph algorithms. Consider a directed graph G defined by: a finite set of vertices $\mathcal{V}$, a set of directed edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, a cost function $w : \mathcal{E} \to \mathbb{R}_{\geq 0}$ for the edges, and a special *start edge* $e_0 \in \mathcal{E}$. The cost of a

path in G is the sum of the costs of the edges in the path. Given $G = (\mathcal{V}, \mathcal{E}, w, e_0)$, the corresponding one-clock initialized SHA $Aut(G)$ is specified by the code in Figure 4. The source and the target vertices of an edge e are denoted by $e[1]$ and $e[2]$, respectively.

$Aut(G)$ where $G = (\mathcal{V}, \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}, w : \mathcal{E} \to \mathbb{R}_{\geq 0}, e_0 \in \mathcal{E})$

Variables:
  $mode \in \mathcal{E}$, **initially** e_0
  $x \in R$, **initially** 0

Actions:
  switch(e,e'), e,$e' \in \mathcal{E}$

Transitions:
  switch(e,e')
    Precondition
      $mode = e \wedge e[2] = e'[1] \wedge x = w(e)$
    Effect
      $mode \leftarrow e'$, $x \leftarrow 0$

Trajectories:
  Trajdef edge($mode$)
  Evolve $d(x) = 1$
  Invariant $x \leq w(mode)$
  Stop when $x = w(mode)$

Fig. 4: Automaton $Aut(G)$ defined by directed graph G

Intuitively, the state of $Aut(G)$ captures the motion of a particle moving with unit speed along the edges of the graph G. The position of the particle is given by the *mode*, which is the edge it resides on, and the value of x, which is its distance from the source vertex of mode. A switch from mode e to mode e' corresponds to the particle arriving at vertex $e[2]$ via edge e, and departing on edge e'. Within edge e the particle moves at unit speed from $e[1]$, where $x = 0$ to $e[2]$, where $x = w(e)$.

The next theorem implies that in order to search for an execution of $Aut(G)$ that maximizes $\mathsf{OPT}(\tau_a)$, it is necessary and sufficient to search over the space of the cycles of G.

Theorem 4.1. *Consider $\tau_a > 0$ and a one-clock initialized SHA $Aut(G)$. $\mathsf{OPT}(\tau_a)$ for $Aut(G)$ is bounded if and only if for all $m > 1$, the cost of any reachable cycle of G with m segments is at least $m\tau_a$.*

Proof. It is easy to see that if there is a cycle of G, $\beta = v_0 e_1 v_1 \dots e_m v_m$, such that the cost $\sum_{i=1}^{m} w(e_i) < m\tau_a$, then $\mathsf{OPT}(\tau_a)$ is unbounded. Since β is a cycle with $v_0 = v_m$, we can construct an execution γ of $Aut(G)$ by concatenating $\beta \frown \beta \frown \beta \dots$, k times. Therefore, the total number of extra mode switches in γ is $S_{\tau_a}(\gamma) = N(\gamma) - \frac{\gamma.ltime}{\tau_a} = km - \frac{k}{\tau_a} \sum_{i=1}^{m} w(e_i) = \frac{k}{\tau_a} (m\tau_a - \sum_{i=1}^{m} w(e_i))$. If $m\tau_a > \sum_{i=1}^{m} w(e_i)$, then the right hand side can be made arbitrarily large by increasing k.

Next, suppose $\mathsf{OPT}(\tau_a)$ is unbounded for $Aut(G)$. We choose N_0 to be larger than the number of vertices $|\mathcal{V}|$ of G. Let β be the shortest execution of $Aut(G)$ with more than N_0 extra switches. Suppose the length of β is l. Since $S_{\tau_a}(\beta) > N_0$, $l - \frac{1}{\tau_a} \sum_{i=1}^{l} w_i > N_0$. Since N_0 is larger than the number of vertices $Aut(G)$, some of the vertices must be repeated in β. That is, β must contain a cycle. Suppose $\beta = \beta_p.\gamma.\beta_s$, where γ is cycle, and let l_1, l_2, l_3 be the lengths of β_p, γ, and β_s,

respectively. Then,

$$l_1 + l_2 + l_3 > N_0 + \frac{1}{\tau_a}\sum_{i=1}^{l_1} w_i + \frac{1}{\tau_a}\sum_{i=1}^{l_2} w_i + \frac{1}{\tau_a}\sum_{i=1}^{l_3} w_i$$

For the sake of contradiction we assume that the cost of the cycle γ, $\sum_{i=1}^{l_2} w_i \geq l_2\tau_a$. Therefore,

$$l_1 + l_3 > N_0 + \frac{1}{\tau_a}\left[\sum_{i=1}^{l_1} w_i + \sum_{i=1}^{l_3} w_i\right] \tag{8}$$

From Equation (8), $S_{\tau_a}(\beta_p \frown \beta_s) > N_0$, and we already know that $\beta_p \frown \beta_s$ is shorter that β, which contradicts our assumption that β is the shortest execution with more than N_0 extra switches. □

Thus, the problem of solving $\mathsf{OPT}(\tau_a)$ for $Aut(G)$ reduces to checking whether G contains a cycle of length m, for any $m > 1$, with cost less than $m\tau_a$. This is the well known mean cost cycle problem for directed graphs and can be solved efficiently using Bellman-Ford algorithm or Karp's minimum mean-weight cycle algorithm [Cormen et al. 1990].

4.2 Linear hysteresis switch

Using Theorem 4.1 and switching simulations we verify the ADT of a linear version of the **HSwitch** automaton of Figure 2 in Section 3.3. For the linear hysteresis switch **LinHSwitch** (shown in Figure 5), we consider monitoring signals generated by linear differential equations. For each $i \in \{1, \ldots, m\}$, $d(\mu_i) = c_i\mu_i$ if $mode = i$, otherwise $d(\mu_i) = 0$. Here the c_i's are positive constants. The switching logic unit implements the same scale independent hysteresis switching as in **HSwitch**.

LinHSwitch(m, h) where $m \in \mathbb{N}, h \in \mathbb{R}_{\geq 0}$

Variables:
- $mode \in \{1, \ldots, m\}$,
  initially $mode = i_0$
- $\mu_i \in R$, $i \in \{1, \ldots, m\}$,
  initialy $\mu_{i_0} = (1+h)C_0$
  $\forall\, i \neq i_0$, $\mu_i = C_0$
- **Derived variables**
  $\mu_{min} = \min_i\{\mu_i\}$

Actions:
- switch(i,j), $i,j \in \{1, \ldots, m\}$

Transitions:
- switch(i,j)
  Precondition
  $mode = i \wedge (1+h)\mu_j \leq \mu_i$
  Effect $mode \leftarrow j$

Trajectories:
- **Trajdef** mode$_i$, $i \in \{1, \ldots, m\}$
  Evolve $d(\mu_i) = c_i\mu_i$
  $d(\mu_j) = 0\ \forall\ j \in \{1, \ldots, m\}, j \neq i$,
  Stop when
  $\exists\ j \in \{1, \ldots, m\}$ **such that** $(1+h)\mu_j \leq \mu_i$

Fig. 5: Linear hysteresis switch

The **LinHSwitch** automaton is not a one-clock initialized SHA. We cannot apply Theorem 4.1 to verify its ADT directly. However, the switching behavior of **LinHSwitch** does not depend on the value of the μ_i's but only on the ratio of $\frac{\mu_i}{\mu_{min}}$, which is always within $[1, (1+h)]$. When **LinHSwitch** is in mode i, all the ratios remain constant, except $\frac{\mu_i}{\mu_{min}}$. The ratio $\frac{\mu_i}{\mu_{min}}$ increases monotonically from

1 to either $(1+h)$ or to $(1+h)^2$, in time $\frac{1}{c_i}\ln(1+h)$ or $\frac{2}{c_i}\ln(1+h)$, respectively. Based on this observation, we will show that there exists a one-clock initialized automaton $\mathcal{B}$, which is equivalent to **LinHSwitch** with respect to ADT.

Consider a graph $G = (\mathcal{V}, \mathcal{E}, w, e_0)$, where:

(1) $\mathcal{V} \subset \{1, (1+h)\}^m$, such that for any $v \in V$, all the m-components are not equal. We denote the i^{th} component of $v \in V$ by $v[i]$.

(2) An edge $(u, v) \in \mathcal{E}$ if and only if, one of the following conditions hold:
(a) There exists $j \in \{1, \ldots, m\}$, such that, $u[j] \neq v[j]$ and for all $i \in \{1, \ldots, m\}$, $i \neq j$, $u[i] = v[i]$. The cost of the edge $w(u, v) := \frac{1}{c_j}\ln(1+h)$ and we define $\zeta(u, v) := j$.
(b) There exists $j \in \{1, \ldots, m\}$ such that $u[j] = 1, v[j] = (1+h)$ and for all $i \in \{1, \ldots, m\}$, $i \neq j$ implies $u[i] = (1+h)$ and $v[i] = 1$. The cost of the edge $w(u, v) := \frac{2}{c_j}\ln(1+h)$ and we define $\zeta(u, v) := j$. The i^{th} component of the source (destination) vertex of edge e is denoted by $e[1][i]$ ($e[2][i]$, respectively).

(3) $e_0 \in \mathcal{E}$, such that $e_0[1][i_0] = (1+h)$ and for all $i \neq i_0$, $e_0[1][i] = 1$.

Let G be the graph of Figure 6. $Aut(G)$ is the one-clock initialized automaton corresponding to **LinHSwitch** with $m = 3$. A typical execution $\alpha = \tau_0, a_1, \tau_1, a_2, \tau_2$ of **LinHSwitch** is as follows: τ_0 is a point trajectory that maps to the state $(mode = 1, [\mu_1 = (1+h)C_0, \mu_2 = C_0, \mu_3 = C_0])$, $a_1 = \mathsf{switch}(1, 3)$, $\tau_1.dom = [0, \frac{1}{c_3}\ln(1+h)]$, $(\tau_1 \downarrow \mu_3)(t) = C_0 e^{c_3 t}$, $a_2 = \mathsf{switch}(3, 2)$, $\tau_2.dom = [0, \frac{2}{c_2}\ln(1+h)]$, $(\tau_2 \downarrow \mu_2)(t) = C_0 e^{c_2 t}$. Note that each edge e of G corresponds to a mode of **LinHSwitch**; this correspondence is captured by the ζ function in the definition of G.

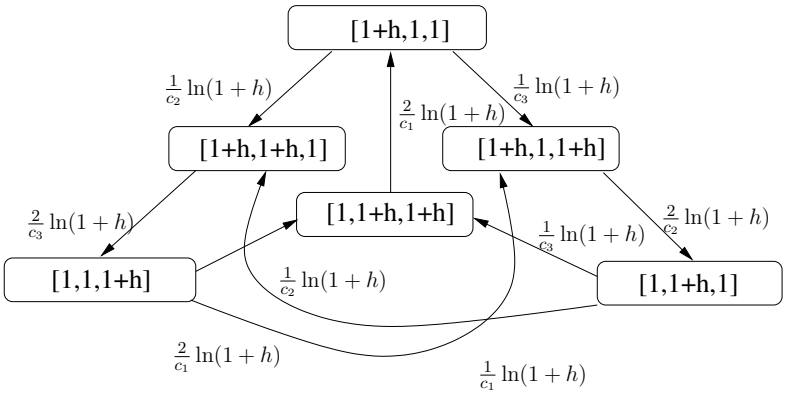

Fig. 6: Graph from which the one-clock initialized abstraction for **LinHSwitch** is obtained.

We define a relation $\mathcal{R}$ on the state spaces on $\mathcal{A} =$ **LinHSwitch** and $\mathcal{B} = Aut(G)$. This relation essentially scales the monitoring signals in **LinHSwitch** by an appropriate factor and equates them with the variable x of $Aut(G)$. The switching pattern of **LinHSwitch** is governed by the multiplicative hysteresis constant h and is independent of this scaling. Indeed, the relation $\mathcal{R}$ will turn out to be a switching simulation relation from $\mathcal{A}$ to $\mathcal{B}$.

Definition 4.2. For any $\mathbf{x} \in Q_{\mathcal{A}}$ and $\mathbf{y} \in Q_{\mathcal{B}}$, $\mathbf{x}\ \mathcal{R}\ \mathbf{y}$ if and only if:

(1) $\zeta(\mathbf{y} \lceil mode) = \mathbf{x} \lceil mode$

(2) For all $j \in \{1, \ldots, n\}$,
(a) $\frac{\mathbf{x}\lceil\mu_j}{\mathbf{x}\lceil\mu_{min}} = e^{c_j(\mathbf{y}\lceil x)}$, if $j = \zeta(\mathbf{y} \lceil mode)$,
(b) $\frac{\mathbf{x}\lceil\mu_j}{\mathbf{x}\lceil\mu_{min}} = (\mathbf{y} \lceil mode)[k][j]$, $k \in \{1, 2\}$.

Part 1 of Definition 4.2 states that if $\mathcal{A}$ is in *mode* j and $\mathcal{B}$ is in mode e, then $\zeta(e) = j$. Part 2 states that for all $j \neq \zeta(e)$, the j^{th} component of $e[1]$ and $e[2]$ are the same, and are equal to μ_j/μ_{min}, and for $j = \zeta(e)$, $\mu_j = \mu_{min}e^{c_j x}$.

Lemma 6.1 states that $\mathcal{R}$ is a switching simulation relation from $\mathcal{A}$ and $\mathcal{B}$ and from $\mathcal{B}$ to $\mathcal{A}$. The proof follows the typical pattern of simulation proofs. We first show that $\mathcal{R}$ is a switching simulation relation from $\mathcal{A}$ to $\mathcal{B}$. This we show by a case analysis that every action and trajectory of automaton $\mathcal{A}$ can be simulated by an execution fragment of $\mathcal{B}$ with at least as many extra switches. The proof of the second part is similar. The complete proof is given in Appendix A.

Lemma 4.3. *$\mathcal{R}$ is a switching simulation relation from $\mathcal{A}$ to $\mathcal{B}$ and from $\mathcal{B}$ to $\mathcal{A}$.*

From Corollary 2.8 it follows that SHA **LinHSwitch** and $Aut(G)$ are ADT-equivalent. As $Aut(G)$ is one-clock initialized SHA, from the results in Section 4.1, we conclude that the ADT properties of $Aut(G)$ and therefore **LinHSwitch** can be verified efficiently by finding the minimum mean cost cycle of G.

For **LinHSwitch** with $m = 3, c_1 = 2, c_2 = 4$, and $c_3 = 5$ we compute the minimum mean-cost cycle. The cost of this cycle, which is also the ADT of this automaton, is $\frac{19}{40}\log(1+h)$. We can also use Theorem 3.7 to get an estimate of the ADT of **LinHSwitch**. If we plug in $\lambda = c_1 = 2$, we get that ADT of this automaton is at least $\frac{1}{6}\log(1+h)$. The discrepancy in the two quantities is because of the fact that the mean-cost cycle analysis uses exact information about the behavior of the monitoring signals whereas the Theorem 3.7 is based on upper and lower bounds given by Equations (5) and (6).

4.3 Initialized SHA

In this section we study ADT properties of *Initialized SHA*. A SHA $\mathcal{A}$ is said to be *initialized* if every action $a \in A$ is associated with two sets $R_a, Pre_a \subseteq Q$, such that $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ is a (mode switching) discrete transition if and only if $\mathbf{x} \in Pre_a$ and $\mathbf{x}' \in R_a$. The set R_a is called the *initialization predicate* of a.

Our next theorem implies that for an initialized SHA $\mathcal{A}$, it is necessary and sufficient to solve $\mathsf{OPT}(\tau_a)$ over the space of the cyclic fragments of $\mathcal{A}$ instead of the larger space of all execution fragments.

Theorem 4.4. *Given $\tau_a > 0$ and initialized SHA $\mathcal{A}$, $\mathsf{OPT}(\tau_a)$ is bounded if and only if $\mathcal{A}$ does not have any cycles with extra switches with respect to τ_a.*

Proof. For simplicity we assume that all discrete transitions of the automaton $\mathcal{A}$ are mode switches and that for any pair of modes i, j, there exists at most one action which can bring about a mode switch from i to j. Existence of a reachable cycle α with extra switches with respect to τ_a is sufficient to show that τ_a is not an ADT for $\mathcal{A}$. This is because by concatenating a sequence of α's, we can construct an execution fragment $\alpha^\frown\alpha^\frown\alpha\ldots$ with an arbitrarily large number of extra switches.

We prove by contradiction that existence of a cycle with extra switches is necessary for making $\mathsf{OPT}(\tau_a)$ unbounded. We assume that $\mathsf{OPT}(\tau_a)$ is unbounded for

$\mathcal{A}$ and that $\mathcal{A}$ does not have any cycles with extra switches. By the definition of OPT, for any constant N_0 there exists an execution that has more than N_0 extra switches with respect to τ_a. Let us choose $N_0 > |\mathcal{P}|^3$. Of all the executions that have more than N_0 extra switches, let $\alpha = \tau_0 a_1 \tau_1 \dots \tau_n$ be a closed execution that has the smallest number of mode switches. From α, we construct $\beta = \tau_0^* a_1 \tau_1^* \dots \tau_n^*$, using the following two rules:

(1) Each τ_i of α is replaced by:

$$\tau_i^* = \arg\min\{\tau.ltime \mid \tau.fstate \in R_{a_i}, \tau.lstate \in Pre_{a_{i+1}}\}.$$

(2) If there exists $i, j \in \{1, \dots, m\}$, such that $a_i = a_j$ and $a_{i+1} = a_{j+1}$, then we make $\tau_i^* = \tau_j^*$.

Claim 4.5. *The sequence β is an execution fragment of $\mathcal{A}$ and $S_{\tau_a}(\beta) > |\mathcal{P}|^3$.*

Proof of claim: We prove the first part of the claim by showing that each application of the above rules to an execution fragment of $\mathcal{A}$ results in another execution fragment. Consider *Rule (1)* and fix i. Since $\tau_i^*.fstate \in R_{a_i}$ and $\tau_{i-1}.lstate \in Pre_{a_i}$, $\tau_{i-1}.lstate \xrightarrow{a_i} \tau_i^*.fstate$. And, since $\tau_i^*.lstate \in Pre_{a_{i+1}}$ and $\tau_{i+1}.fstate \in R_{a_{i+1}}$, we know that $\tau_i^*.lstate \xrightarrow{a_{i+1}} \tau_{i+1}.fstate$. Now for *Rule (2)*, we assume there exist i and j such that the hypothesis of the rule holds and suppose $\tau_j^* = \tau_i^* = \tau_i$. We know that even if $\tau_j^* \neq \tau_j$, the first states of both are in R_{a_j} and the last states are in $Pre_{a_{j+1}}$. Therefore, a_j matches up the states of τ_{j-1} and τ_j^* and likewise a_{j+1} matches the states of τ_j^* and τ_{j+1}.

The second part of the claim follows from the fact that each trajectory τ_i is replaced by the shortest trajectory τ_i^* from the initialization set of the previous transition R_{a_i} to the guard set of the next transition $Pre_{a_{i+1}}$. That is, for each i, $0 < i < n$, $\tau_i^*.ltime \leq \tau_i.ltime$ and therefore $\beta.ltime \leq \alpha.ltime$ and $S_{\tau_a}(\beta) > N_0 > |\mathcal{P}|^3$.

Since $N(\beta) > |\mathcal{P}|^3$, there must be a sequence of 3 consecutive modes that appear multiple times in β. That is, there exist $i, j \in \{1, \dots, m\}$, and $p, q, r \in \mathcal{P}$, such that $\tau_i^*.fstate \lceil mode = \tau_j^*.fstate \lceil mode = p$, $\tau_{i+1}^*.fstate \lceil mode = \tau_{j+1}^*.fstate \lceil mode = q$, and $\tau_{i+2}^*.fstate \lceil mode = \tau_{j+2}^*.fstate \lceil mode = r$. Then, from *Rule (2)* we know that $\tau_{i+1}^* = \tau_{j+1}^*$. In particular, $\tau_{i+1}^*.fstate = \tau_{j+1}^*.fstate$, that is, we can write $\beta = \beta_p \frown \gamma \frown \beta_s$, where γ is a cycle. Then we have the following:

$$N(\beta_p) + N(\gamma) + N(\beta_s) > N_0 + \beta_p.ltime/\tau_a + \gamma.ltime/\tau_a + \beta_s.ltime/\tau_a$$
$$N(\beta_p) + N(\beta_s) + S_{\tau_a}(\gamma) > N_0 + \beta_p.ltime/\tau_a + \beta_s.ltime/\tau_a$$
$$N(\beta_p \frown \beta_s) > N_0 + \beta_p \frown \beta_s.ltime/\tau_a \qquad [\beta_p.lstate = \beta_s.fstate]$$

The last step follows from the assumption that $S_{\tau_a}(\gamma) \leq 0$. Therefore, we have $S_{\tau_a}(\beta_p \frown \beta_s) > N_0$ which contradicts our assumption that β has the smallest number of mode switches among all the executions that have more than N_0 extra switches with respect to τ_a. □

The following corollary allows us to limit the search for cycles with extra switches to cycles with at most $|\mathcal{P}|^3$ mode switches. It is proved by showing that any cycle with extra switches that has more than $|\mathcal{P}|^3$ mode switches can be decomposed into two smaller cycles, one of which must also have extra switches.

Corollary 4.6. *If initialized SHA $\mathcal{A}$ has a cycle with extra switches, then it has a cycle with extra switches that has fewer than $|\mathcal{P}|^3$ mode switches.*

PROOF. Follows from the last part of the proof of Theorem 4.4. □

4.4 MILP formulation of OPT(τ_a)

A SHA is *rectangular* if the differential equations in the state models have constant right hand sides, and the precondition and the initialization predicates (restricted to the set of continuous variables) are polyhedral. We use the above results to solve the ADT verification problem for rectangular initialized SHA with Mixed Integer Linear Programming (MILP). Figure 7 shows the specification of a generic initialized rectangular SHA $\mathcal{A}$. The automaton $\mathcal{A}$ has a single discrete variable called *mode* which takes values in the index set $\mathcal{P} = \{1, \ldots, N\}$, and a continuous variable vector $\mathbf{x} \in \mathbb{R}^n$. For any $i, j \in \mathcal{P}$, the action that changes the mode from i to j is called switch(i, j). The precondition and the initialization predicates of this action are given by sets of linear inequalities on the continuous variables, represented by: $G[i, j]\mathbf{x} \leq g[i, j]$ and $R[i, j]\mathbf{x} \leq r[i, j]$, respectively, where $G[i, j]$ and $R[i, j]$ are constant matrices with N columns and $g[i, j], r[i, j]$ are constant vectors.

Rectangular$(\mathcal{P}, G, A, R, q, a, r, c)$

Variables:
  mode $\in \mathcal{P}$, **initially** p
  $\mathbf{x} \in R^n$, **initially** $\mathbf{x}_0$

Actions
  switch(p,q), $p,q \in \mathcal{P}$

Transitions:
  switch(p,q)
    Precondition
      mode $= p \wedge G[p,q]\mathbf{x} \leq g[p,q]$
    Effect
      mode $\leftarrow q$
      $\mathbf{x} \leftarrow \mathbf{x}'$ such that $R[p,q]\mathbf{x}' \leq r[p,q]$

Trajectories:
  Trajdef mode(p)
  Invariant $A[p]\mathbf{x} \leq a[p]$
  Evolve $d(\mathbf{x}) = c[p]$

Fig. 7: Generic rectangular initialized SHA with parameters $\mathcal{P}, G, A, R, q, a, r, c$.

For each mode $i \in \mathcal{P}$, the invariant is stated in terms of linear inequalities of the continuous variables $A[i]\mathbf{x} \leq a[i]$, where $A[i]$ is a constant matrix with n columns and $a[i]$ is a constant vector. The evolve clause is given by a single differential equation $d(\mathbf{x}) = c[i]$, where $c[i]$ is a constant vector.

We describe a MILP formulation MOPT(K, τ_a) for finding a cyclic execution with K mode switches that maximizes the number of extra switches with respect to τ_a. If the optimal value is positive, then the optimal solution represents a cycle with extra switches with respect to τ_a and we conclude from Corollary 4.6 that τ_a is not an ADT for $\mathcal{A}$. On the other hand, if the optimal value is not positive, then we conclude that there are no cycles with extra switches of length K. To verify ADT of $\mathcal{A}$, we solve a sequence of MOPT(K, τ_a)'s with $K = 2, \ldots, |\mathcal{P}|^3$. If the optimal values are not positive for any of these, then we conclude that τ_a is an ADT for $\mathcal{A}$. By adding extra variables and constraints we are able to formulate a single MILP that maximizes the extra switches over all cycles with K or less mode switches,

but for simplicity of presentation we discuss $\mathsf{MOPT}(K, \tau_a)$ instead of this latter formulation. The following are the decision variables for $\mathsf{MOPT}(K, \tau_a)$.

—$\mathbf{x}_u \in \mathbb{R}^n$, $u \in \{0 \ldots, K\}$, value of continuous variables
—$t_u \in \mathbb{R}$, $u \in \{0, 2, 4, \ldots, K\}$, length of u^{th} trajectory
—$m_{uj} = \begin{cases} 1, & \text{if mode over } u^{th} \text{ trajectory is } j \\ 0, & \text{otherwise.} \end{cases}$ for each $u \in \{0, 2, \ldots, K\}$, $j \in \{1, \ldots, N\}$
—$p_{ujk} = \begin{cases} 1, & \text{if mode over } (u-1)^{st} \text{ trajectory is } j \text{ and over } (u+1)^{st} \text{ trajectory is } k \\ 0, & \text{otherwise.} \end{cases}$ for each $u \in \{0, 2, 4, \ldots, K\}, j, k \in \{1, \ldots, N\}$

The objective function and the constraints are shown in Figure 8. In $\mathsf{MOPT}(K, \tau_a)$, an execution fragment with K mode switches is represented as a sequence $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_K$ of K valuations for the continuous variables. For each even u, $\mathbf{x}_u$ goes to $\mathbf{x}_{u+1}$ by a trajectory of length t_u. If this trajectory is in mode j, for some $j \in \{1, \ldots, N\}$, then $m_{uj} = 1$, else $m_{uj} = 0$. For each odd u, $\mathbf{x}_u$ goes to $\mathbf{x}_{u+1}$ by a discrete transition. If this transition is from mode j to mode k, for some $j, k \in \{1, \ldots, N\}$, then $p_{ujk} = 1$, else $p_{ujk} = 0$. These constraints are specified by Equation (9) in Figure 8. For each odd u, Constraints (11) and (12) ensure that $(\mathbf{x}_u, \mathsf{switch}(j, k), \mathbf{x}_{u+1})$

$$\text{Objective function:} \quad S_{\tau_a} : \frac{K}{2} - \frac{1}{\tau_a} \sum_{u=0,2,\ldots}^{K} t_u$$

$$\text{Mode:} \quad \forall\, u \in \{0, 2, \ldots, K\},\ \sum_{j=1}^{N} m_{uj} = 1 \ \textit{and} \ \forall\, u \in \{1, 3, \ldots, K-1\},\ \sum_{j=1}^{N} \sum_{k=1}^{N} p_{ujk} = 1 \tag{9}$$

$$\text{Cycle:} \quad \mathbf{x}_0 = \mathbf{x}_K \text{ and } \forall\, j \in \{1, \ldots, N\}, m_{0j} = m_{Kj} \tag{10}$$

$$\text{Preconds:} \quad \forall\, u \in \{1, 3, \ldots, K-1\},\ \sum_{j=1}^{N} \sum_{k=1}^{N} G[j,k].p_{ujk}.\mathbf{x}_u \leq \sum_{j=1}^{N} \sum_{k=1}^{N} p_{ujk}.g[j,k] \tag{11}$$

$$\text{Initialize:} \quad \forall\, u \in \{1, 3, \ldots, K-1\},\ \sum_{j=1}^{N} \sum_{k=1}^{N} R[j,k].p_{ujk}.\mathbf{x}_{u+1} \leq \sum_{j=1}^{N} \sum_{k=1}^{N} p_{ujk}.r[j,k] \tag{12}$$

$$\text{Invariants:} \quad \forall\, u \in \{0, 2, \ldots, K\},\ \sum_{j=1}^{N} A[j].m_{uj}.\mathbf{x}_u \leq \sum_{j=1}^{N} m_{uj}.a[j] \tag{13}$$

$$\text{Evolve:} \quad \forall\, u \in \{0, 2 \ldots, K\},\ \mathbf{x}_{u+1} = \mathbf{x}_u + \sum_{j=1}^{N} c[j].m_{uj}.t_u \tag{14}$$

Fig. 8: The objective function and the linear and integral constraints for $\mathsf{MOPT}(K, \tau_a)$

is a valid mode switching transition. These constraints simplify to the inequalities $G[j,k]\mathbf{x}_u \leq g[j,k]$ and $R[j,k]\mathbf{x}_{u+1} \leq r[j,k]$ which correspond to the precondition and the initialization conditions on the pre and the post-state of the transition. For each even u, $\mathbf{x}_u$ evolves to $\mathbf{x}_{u+1}$ through a trajectory in some mode, say j. Constraint (13) ensures that $\mathbf{x}_u$ satisfies the invariant of mode j described by the inequality $A[j]\mathbf{x}_u \leq a[j]$. An identical constraint for $\mathbf{x}_{u+1}$ is written by replacing $\mathbf{x}_u$ with $\mathbf{x}_{u+1}$ in (13). Since the differential equations have constant right hand sides and the invariants describe polyhedra in $\mathbb{R}^n$, the above conditions ensure

that all the intermediate states in the trajectory satisfy the mode invariant. Equation (14) ensures that, for each even u, $\mathbf{x}_u$ evolves to $\mathbf{x}_{u+1}$ in t_u time according to the differential equation $d(\mathbf{x}) = c[j]$.

Some of these constrains involve nonlinear terms. Using the "big M" method [Williams 1990] we can linearize these equation and inequalities. For example, $m_{uj}\mathbf{x}_u$ in (13) is the product of real variable $\mathbf{x}_u$ and boolean variable m_{uj}. We linearize it by replacing $m_{uj}\mathbf{x}_u$ with $\mathbf{y}_u$, and adding the following linear inequalities: $\mathbf{y}_u \geq m_{uj}\delta$, $\mathbf{y}_u \leq m_{uj}\Delta$, $\mathbf{y}_u \leq \mathbf{x}_u - (1 - m_{uj})\delta$, and $\mathbf{y}_u \geq \mathbf{x}_u - (1 - m_{uj})\Delta$, where δ and Δ are the lower and upper bounds on the values of $\mathbf{x}_u$.

4.5 Thermostat

We use the MILP technique together with switching simulation relations to verify the ADT of a thermostat with nondeterministic switches. The model of the thermostat is the **Thermostat** SHA (see Figure 9 *Left*). It has two modes l_0, l_1, two continuous variables x and z, and real parameters $h, K, \theta_1, \theta_2, \theta_3, \theta_4$, where $0 < \theta_1 < \theta_2 < \theta_3 < \theta_4 < h$. In l_0 mode the heater is off and the temperature x decreases according to the differential equation $d(x) = -Kx$. While the temperature x is between θ_2 and θ_1, the on action *must* occur. As an effect of the on action, the mode changes to l_1. In mode l_1, the heater is on and the x rises according to the $d(x) = K(h - x)$, and while x is between θ_3 and θ_4, the off action must occur. The continuous variable z measures the total time spent in mode l_1.

SHA **Thermostat** is neither initialized nor rectangular; however, there is a rectangular initialized SHA **Approx**, such that **Thermostat** $\geq_{ADT}$ **Approx**. Consider the SHA **Approx** of Figure 9 (*Right*) with parameters L_0 and L_1. Automaton **Approx** has a clock t and two modes l_0 and l_1, in each of which t increases at a unit rate. When t reaches L_i in mode l_i, a switch to the other mode *may* occur and if it does then t is set to zero. We define a relation $\mathcal{R}$ on the state spaces of **Thermostat** and **Approx** such that with appropriately chosen values of L_0 and L_1, **Approx** captures the fastest switching behavior of **Thermostat**.

Definition 4.7. For any $\mathbf{x} \in Q_{\mathbf{Thermostat}}$ and $\mathbf{y} \in Q_{\mathbf{Approx}}$, $\mathbf{x}\ \mathcal{R}\ \mathbf{y}$ if and only if:
(1) $\mathbf{x} \lceil mode = \mathbf{y} \lceil mode$, and
(2) if $\mathbf{x} \lceil mode = l_0$ then $\mathbf{y} \lceil t \geq \frac{1}{k} \ln \frac{\theta_3}{\mathbf{x} \lceil x}$ else $\mathbf{y} \lceil t \geq \frac{1}{k} \ln \left(\frac{h - \theta_2}{h - \mathbf{x} \lceil x} \right)$.

Lemma 4.8. *If we set* $L_0 = \frac{1}{k} \ln \frac{\theta_3}{\theta_2}$ *and* $L_1 = \frac{1}{k} \ln \frac{h - \theta_2}{h - \theta_3}$, *then the relation* $\mathcal{R}$ *is a switching simulation relation from* **Thermostat** *to* **Approx**.

The proof of this lemma is like that of Lemma 6.1. We show that every reachable state of **Thermostat** is related to some state of **Approx** and that every action and trajectory of **Thermostat** can be emulated by an execution fragment of **Approx** with no fewer switches. Lemma 4.8 implies that **Thermostat** $\geq_{ADT}$ **Approx**, that is, for any $\tau_a > 0$ if τ_a is an ADT for **Approx** then τ_a is also an ADT for **Thermostat**. Since **Approx** is rectangular and initialized, we can use the MILP technique to check any ADT property of **Approx**.

We formulated the $\mathsf{MOPT}(K, \tau_a)$ for automaton **Approx** and used the GNU Linear Programming Kit [GNU] to solve it. Solving for $K = 4, L_0 = 40, L_1 = 15$, and $\tau_a = 25, 27, 28$, we get optimal costs $-0.4, -4.358E^{-13} (\approx 0)$ and 0.071, respectively. We conclude that the ADT of **Approx** is $\geq 25, \geq 27$, and < 28. Since

Thermostat$(\theta_1, \theta_2, \theta_3, \theta_4, C, h)$
where $\theta_1, \theta_2, \theta_3, \theta_4, C, h \in \mathbb{R}$

Variables:
  $mode \in \{l_0, l_1\}$, **initially** l_0
  $x, z \in R$, **initially** $x = \theta_4, z = 0$

Actions
  on, off

Transitions:
  on **Precondition**
    $mode = l_0 \wedge x \leq \theta_2$
  Effect $mode \leftarrow l_1$

  off **Precondition**
    $mode = l_1 \wedge x \geq \theta_3$
  Effect $mode \leftarrow l_0$

Trajectories:
  Trajdef l_0
  Evolve $d(x) = -Cx$; $d(z) = 0$
  Invariant $x \geq \theta_1$ **Stop when** $x = \theta_1$

  Trajdef l_1
  Evolve $d(x) = C(h-x)$; $d(z) = 1$
  Invariant $x \leq \theta_4$ **Stop when** $x = \theta_4$

Approx(L_0, L_1), *where* $l_1, l_0 \in \mathbb{R}$

Variables:
  $mode \in \{l_0, l_1\}$, **initially** l_0
  $r \in R$, **initially** $r = L_1$

Actions
  switchto$_i$, $i \in \{0,1\}$

Transitions:
  switchto$_1$
  Precondition
    $mode = l_0 \wedge r \geq L_0$
  Effect $mode \leftarrow l_1$, $r \leftarrow 0$

  switchto$_0$
  Precondition
    $mode = l_1 \wedge r \geq L_1$
  Effect $mode \leftarrow l_0$, $r \leftarrow 0$

Trajectories:
  Trajdef always
  Evolve $d(r) = 1$

Fig. 9: **Thermostat** SHA and its abstraction **Approx** rectangular initialized SHA.

Approx $\geq_{ADT}$ **Thermostat**, we conclude that the ADT of the thermostat is no less than 27.

Remark. For finding counterexample execution fragments for the proposed ADT properties, the MILP approach can be applied to non-initialized rectangular SHA as well. In such applications, the necessity part of Theorem 4.4 does not hold and therefore from the failure to find a counterexample we cannot conclude that the automaton satisfies the ADT property in question.

5. CONCLUSIONS

In this paper we have presented two methods for proving ADT properties of hybrid systems. Stability of a hybrid system is guaranteed it its individual modes are stable and if it has the appropriate ADT property.

The first method transforms the given automaton by adding history variables, such that the transformed automaton satisfies an invariant property if and only if the original automaton has the ADT property. To prove the resulting invariant properties, we appeal to the large body of tools available for proving invariants for hybrid systems. This method for proving ADT properties is applicable to any hybrid system; however, automatic verification is only possible for the class of systems for which model checking is decidable. For hybrid systems outside this class, semi-automatic proofs can be carried out with the aid of theorem provers.

The second method relies on solving optimization problems. We have shown

that for the class of rectangular initialized hybrid automata, ADT properties can be verified or counterexample executions can be found automatically by solving mixed integer linear programs. For non-initialized hybrid automata, the solution of the optimization problem can give executions that serve as counterexamples to the ADT property in question, but for this class the method is not complete. The two methods can be combined to find the ADT of a given rectangular hybrid automaton.

We have defined equivalence of hybrid automata with respect to switching speed. We proposed a new kind of simulation relation, namely switching simulation, which gives a sufficient condition for establishing that two hybrid automata are equivalent with respect to ADT.

In the future we want to explore other properties of hybrid automata that can be formulated as search problems over the space of executions and can be solved using optimization. In this paper we examined internal stability only; however, using the explicit external variables of the HIOA framework, we would like to study input-output properties of hybrid systems as well. Another direction of future research is to extend these techniques to stochastic hybrid systems, by combining the probabilistic I/O automaton models of [Canetti et al. 2006; Mitra and Lynch 2006] with stability results for stochastic switched systems from [Chatterjee and Liberzon 2004].

Acknowledgment

We would like to thank Debasish Chatterjee for giving us many useful comments and suggestions on this paper.

REFERENCES

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T. A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. 1995. The algorithmic analysis of hybrid systems. *Theoretical Computer Science 138,* 1, 3–34. 2, 4

Alur, R., Henzinger, C. C. T. A., and Ho., P. H. 1993. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, Eds. Lecture Notes in Computer Science, vol. 736. Springer-Verlag, 209–229. 13

Branicky, M. 1995. Studies in hybrid systems: modeling, analysis, and control. Ph.D. thesis, MIT, Cambridge, MA. 4

Branicky, M. 1998. Multiple lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on Automatic Control 43*, 475–482. 2

Branicky, M., Borkar, V., and Mitter, S. 1998. A unified framework for hybrid control: model and optimal control theory. *IEEE Transactions on Automatic Control 43,* 1, 31–45. 4

Canetti, R., Cheung, L., Kaynar, D., Liskov, M., Lynch, N., Pereira, O., and Segala, R. 2006. Task-structured probabilistic I/O automata. Tech. Rep. MIT-CSAIL-TR-2006-023, Massachusetts Institute of Technology, Cambridge, MA. March. 26

Chatterjee, D. and Liberzon, D. 2004. On stability of stochastic switched systems. In *Proceedings of the 43rd Conference on Decision and Control*. Paradise Island, Bahamas, 4125–4127. 26

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. 1990. *Introduction to Algorithms*. MIT Press/McGraw-Hill. 18

Frehse, G. 2005a. Compositional verification of hybrid systems using simulation relations. Ph.D. thesis, Radbout Universiteit Nijmegen. 9

Frehse, G. 2005b. Phaver: Algorithmic verification of hybrid systems past hytech. In *HSCC*, M. Morari and L. Thiele, Eds. Lecture Notes in Computer Science, vol. 3414. Springer, 258–273. 3

GNU. GLPK - GNU linear programming kit. Available from http://www.gnu.org/directory/libs/glpk.html. 24

Heitmeyer, C. and Lynch, N. 1994. The generalized railroad crossing: A case study in formal verification of real-time system. In *Proceedings of the 15th IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, San Juan, Puerto Rico. 2

Henzinger, T. A., Ho, P.-H., and Wong-Toi, H. 1997. Hytech: A model checker for hybrid systems. In *Computer Aided Verification (CAV '97)*. Lecture Notes in Computer Science, vol. 1254. 460–483. 3, 13

Henzinger, T. A. and Majumdar, R. 2000. Symbolic model checking for rectangular hybrid systems. In *Proceedings of the Sixth International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*. Lecture Notes in Computer Science 1785, Springer-Verlag, 142–156. 2

Hespanha, J., Liberzon, D., and Morse, A. 2003. Hysteresis-based switching algorithms for supervisory control of uncertain systems. *Automatica 39*, 263–272. 13

Hespanha, J. and Morse, A. 1999. Stability of switched systems with average dwell-time. In *Proceedings of 38th IEEE Conference on Decision and Control*. 2655–2660. 2, 8

Kaynar, D., Lynch, N., Mitra, S., and Garland, S. 2005. *TIOA Language*. MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA. 6

Kaynar, D. K., Lynch, N., Segala, R., and Vaandrager, F. 2005. *The Theory of Timed I/O Automata*. Synthesis Lectures on Computer Science. Morgan Claypool. Also available as Technical Report MIT-LCS-TR-917. 5, 10

Khalil, H. K. 2002. *Nonlinear Systems*, 3rd ed. Prentice Hall, New Jersey. 3, 7

Kurzhanski, A. B. and Varaiya, P. 2000. Ellipsoidal techniques for reachability analysis. In *HSCC*. 202–214. 2

Liberzon, D. 2003. *Switching in Systems and Control*. Systems and Control: Foundations and Applications. Birkhauser, Boston. 2, 4, 8, 13, 16

Lim, H., Kaynar, D., Lynch, N., and Mitra, S. 2005. Translating timed i/o automata specifications for theorem proving in pvs. In *Proceedings of Formal Modelling and Analysis of Timed Systems (FORMATS'05)*. Number 3829 in LNCS. Springer, Uppsala, Sweden. 9

Livadas, C., Lygeros, J., and Lynch, N. A. 1999. High-level modeling and analysis of TCAS. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99),Phoenix, Arizona*. 115–125. 2

Lynch, N. 1996. A three-level analysis of a simple acceleration maneuver, with uncertainties. In *Proceedings of the Third AMAST Workshop on Real-Time Systems*. World Scientific Publishing Company, Salt Lake City, Utah, 1–22. 9

Lynch, N., Segala, R., and Vaandrager, F. 2003. Hybrid I/O automata. *Information and Computation 185,* 1 (August), 105–157. 2, 3, 4, 5

Lynch, N. and Vaandrager, F. 1996. Forward and backward simulations - part II: Timing-based systems. *Information and Computation 128,* 1 (July), 1–25. 9

Mitchell, I. and Tomlin, C. 2000. Level set methods for computation in hybrid systems. In *HSCC*. 310–323. 2

Mitra, S. and Archer, M. 2005. PVS strategies for proving abstraction properties of automata. *Electronic Notes in Theoretical Computer Science 125,* 2, 45–65. 4, 9

Mitra, S. and Lynch, N. 2006. Probabilistic timed I/O automata with continuous state spaces. Submitted for review. Available from http://theory.lcs.mit.edu/~mitras/research/CONCUR06_06.pdf. 26

Mitra, S., Wang, Y., Lynch, N., and Feron, E. 2003. Safety verification of model helicopter controller using hybrid Input/Output automata. In *HSCC'03, Hybrid System: Computation and Control*. Prague, the Czech Republic. 2

Morse, A. S. 1996. Supervisory control of families of linear set-point controllers, part 1: exact matching. *IEEE Transactions on Automatic Control 41*, 1413–1431. 2, 8

Owre, S., Rajan, S., Rushby, J., Shankar, N., and Srivas, M. 1996. PVS: Combining specification, proof checking, and model checking. In *Computer-Aided Verification, CAV '96*, R. Alur and T. A. Henzinger, Eds. Number 1102 in Lecture Notes in Computer Science. Springer-Verlag, New Brunswick, NJ, 411–414. 3

Prajna, S. and Jadbabaie, A. 2004. Safety verification of hybrid systems using barrier certificates. In *HSCC*. Vol. 2993. 2

van der Schaft, A. and Schumacher, H. 2000. *An Introduction to Hybrid Dynamical Systems*. Springer, London. 2

Weinberg, H. B. and Lynch, N. 1996. Correctness of vehicle control systems – a case study. In *17th IEEE Real-Time Systems Symposium*. Washington, D. C., 62–72. 9

Williams, H. 1990. *Model building in mathematical programming*. J. Wiley, New York. third edition. 24

6. APPENDIX A

Lemma 6.1. *$\mathcal{R}$ is a switching simulation relation from $\mathcal{A}$ to $\mathcal{B}$.*

Proof. We first show that $\mathcal{R}$ is a switching simulation relation from $\mathcal{A}$ to $\mathcal{B}$. At a given state $\mathbf{x}$ of $\mathcal{A}$, we define $i \in \{1, \ldots, m\}$ to be the *unique minimum at* $\mathbf{x}$, if $\min_j\{\mathbf{x} \lceil \mu_j\}$ is unique and $\mu_i = \arg\min_j\{\mathbf{x} \lceil \mu_j\}$. $\mathcal{A}$ has a unique start state and it is easy to see that it is related to all the start states of $\mathcal{B}$. Next we show by cases that given any state $\mathbf{x} \in Q_{\mathcal{A}}, \mathbf{y} \in Q_{\mathcal{B}}$, $\mathbf{x}\ \mathcal{R}\ \mathbf{y}$, and an execution fragment α of $\mathcal{A}$ starting from $\mathbf{x}$ and consisting of either a single action or a single trajectory, there exists a corresponding execution fragment β of $\mathcal{B}$, starting from $\mathbf{y}$ that satisfies the conditions required for $\mathcal{R}$ to be a switching simulation relation.

(1) α is a $(\mathbf{x}, \mathsf{switch}(i,j), \mathbf{x}')$ transition of $\mathcal{A}$ and i is not the unique minimum at $\mathbf{x}$ and j is not unique minimum at $\mathbf{x}'$.
We choose β to be $(\mathbf{y}, \mathsf{switch}(e, e'), \mathbf{y}')$ action of $\mathcal{B}$, where e and e' are determined by the following rules:

$$e[1][i] = 1, e[2][i] = 1 + h, \forall k, k \neq i, e[1][k] = e[2][k] = \frac{\mathbf{x} \lceil \mu_k}{\mathbf{x} \lceil \mu_{min}} \tag{15}$$

$$e'[1][j] = 1, e'[2][j] = 1 + h, \forall k, k \neq j, e'[1][k] = e'[2][k] = \frac{\mathbf{x}' \lceil \mu_k}{\mathbf{x}' \lceil \mu_{min}} \tag{16}$$

We have to show that $\mathsf{switch}(e, e')$ is enabled at $\mathbf{y}$; this involves showing that the three conjuncts in the precondition of the switch action of $\mathcal{B}$ are satisfied at $\mathbf{y}$. First of all, since $\mathbf{x}\ \mathcal{R}\ \mathbf{y}$ we know that $\zeta(\mathbf{y} \lceil mode) = \mathbf{x} \lceil mode = i$. Further, i is not a unique minimum at $\mathbf{x}$, so from the definition of the edges of G it follows that:

$$(\mathbf{y} \lceil mode)[1][i] = 1, (\mathbf{y} \lceil mode)[2][i] = i + h,$$
$$\forall k, k \neq i, (\mathbf{y} \lceil mode)[1][k] = (\mathbf{y} \lceil mode)[2][k] = \frac{\mathbf{x} \lceil \mu_k}{\mathbf{x} \lceil \mu_{min}} \tag{17}$$

Comparing Equations (17) and (23) we conclude that $\mathbf{y} \lceil mode = e$.
Secondly, using the definitions of e, e' and $\mathcal{R}$ it follows that:

$$e[2][i] = 1 + h = \frac{\mathbf{x} \lceil \mu_i}{\mathbf{x} \lceil \mu_{min}} = \frac{\mathbf{x}' \lceil \mu_i}{\mathbf{x}' \lceil \mu_{min}} = e'[1][i] \tag{18}$$

The second equality holds because $\mathsf{switch}(i,j)$ is enabled at $\mathbf{x}$. The third equality follows from the fact that the $\mathsf{switch}(i,j)$ transition of $\mathcal{A}$ does not alter the value of the μ_k's. Likewise, we have:

$$e[2][j] = \frac{\mathbf{x} \lceil \mu_j}{\mathbf{x} \lceil \mu_{min}} = \frac{\mathbf{x}' \lceil \mu_j}{\mathbf{x}' \lceil \mu_{min}} = 1 = e'[1][j] \tag{19}$$

$$\forall k, k \neq j, k \neq i, e[2][k] = \frac{\mathbf{x} \lceil \mu_k}{\mathbf{x} \lceil \mu_{min}} = \frac{\mathbf{x}' \lceil \mu_k}{\mathbf{x}' \lceil \mu_{min}} = e'[1][k] \tag{20}$$

Combining Equations (18),(18) and (20) it follows that $e[2] = e'[1]$.
Finally, from the switching simulation relation $\mathcal{R}$, we know that $\mathbf{y} \lceil x = \frac{1}{c_i} \ln \frac{\mathbf{x} \lceil \mu_i}{\mathbf{x} \lceil \mu_{min}} = \frac{1}{c_i} \ln(1+h)$. And since $\zeta(e) = i$ and i is not the unique minimum at $\mathbf{x}$, from the definition of the edge costs of G it follows that $\mathbf{y} \lceil x = w(e)$. Thus, we have shown that $\mathsf{switch}(e, e')$ is indeed enabled at $\mathbf{y}$.
Next, we have to show that $\mathbf{x}' \mathcal{R} \mathbf{y}'$. First of all, $\mathbf{x}' \lceil mode = j$ and $\mathbf{y}' \lceil mode = e'$ from the effect parts of the $\mathsf{switch}(i, j)$ and $\mathsf{switch}(e, e')$ actions, respectively. Also, $\zeta(e') = j$ from Equation (23). It follows that $\mathbf{x}' \lceil mode = \zeta(\mathbf{y}' \lceil mode)$. Secondly, $\frac{\mathbf{x}' \lceil \mu_j}{\mathbf{x}' \lceil \mu_{min}} = \frac{\mathbf{x} \lceil \mu_j}{\mathbf{x} \lceil \mu_{min}} = 1$, from the precondition of $\mathsf{switch}(i, j)$. Since $\mathbf{y}' \lceil x = 0$ it follows that $\frac{\mathbf{x} \lceil \mu_j}{\mathbf{x} \lceil \mu_{min}} = e^{c_j \mathbf{y}' \lceil x}$. Finally, for all $k \neq j$, again from Equation (23) it follows that $\frac{\mathbf{x}' \lceil \mu_k}{\mathbf{x}' \lceil \mu_{min}} = e'[1][k] = e'[2][k]$.

(2) α is a $(\mathbf{x}, \mathsf{switch}(i, j), \mathbf{x}')$ transition of $\mathcal{A}$ and i is the unique minimum at $\mathbf{x}$ and j is not unique minimum at $\mathbf{x}'$.
We choose β to be $(\mathbf{y}, \mathsf{switch}(e, e'), \mathbf{y}')$ action of $\mathcal{B}$, where e and e' are determined by the following rules:

$$e[1][i] = 1, e[2][i] = 1 + h, \forall k, k \neq i, e[1][k] = (1 + h), e[2][k] = 1 \tag{21}$$

$$e'[1][j] = 1, e'[2][j] = 1 + h, \forall k, k \neq j, e'[1][k] = e'[2][k] = \frac{\mathbf{x}' \lceil \mu_k}{\mathbf{x}' \lceil \mu_{min}} \tag{22}$$

The rest of the proof is similar to that of case 1.

(3) α is a $(\mathbf{x}, \mathsf{switch}(i, j), \mathbf{x}')$ transition of $\mathcal{A}$ and i is not the unique minimum at $\mathbf{x}$ and j is the unique minimum at $\mathbf{x}'$.
We choose β to be $(\mathbf{y}, \mathsf{switch}(e, e'), \mathbf{y}')$ action of $\mathcal{B}$, where e and e' are determined by the following rules:

$$e[1][i] = 1, e[2][i] = 1 + h, \forall k, k \neq i, e[1][k] = e[2][k] = \frac{\mathbf{x} \lceil \mu_k}{\mathbf{x} \lceil \mu_{min}}$$
$$e'[1][j] = 1, e'[2][j] = 1 + h, \forall k, k \neq j, e'[1][k] = (1 + h), e'[2][k] = 1$$

The rest of the proof is similar to that of case 1.

(4) α is a closed trajectory τ of $\mathcal{A}$ with $(\tau \downarrow mode)(0) = i$ for some $i \in \mathcal{P}$, such that i is not unique minimum at $\tau.fstate$.
We choose β to be the trajectory τ' of $\mathcal{B}$ with $\tau'.dom = \tau.dom$ determined by the following rules. Let $\mathbf{x} = \tau.fstate, \mathbf{x}' = \tau.lstate, \mathbf{y} = \tau'.fstate$ and $\mathbf{y}' = \tau'.lstate$.

$$(\mathbf{y} \lceil mode)[1][i] = 1, (\mathbf{y} \lceil mode)[2][i] = 1 + h,$$
$$\forall k, k \neq i, (\mathbf{y} \lceil mode)[1][k] = (\mathbf{y} \lceil mode)[1][k] = \frac{\mathbf{x} \lceil \mu_i}{\mathbf{x} \lceil \mu_{min}},$$
$$\forall\, t \in \tau'.dom, (\tau' \downarrow x)(t) = \frac{1}{c_i} \ln \frac{\mathbf{x} \lceil \mu_i}{\mathbf{x} \lceil \mu_{min}} + t \tag{23}$$

We first show that τ' is a valid trajectory of $\mathcal{B}$. First of all, it is easy to check that τ' satisfies the constant differential equation $d(x) = 1$ and that the $mode$ of $\mathcal{B}$ remains constant. Next, we show that τ' satisfies the stopping condition

"$x = w(mode)$". Suppose there exists $t \in \tau'.dom$ such that $(\tau \lceil x)(t) = w(\mathbf{x} \lceil mode)$, then $t = w(\mathbf{x} \lceil mode) - (\mathbf{y} \lceil x)$. Then,

$$\begin{aligned}
\mathbf{x}' \lceil \mu_i &= \mathbf{x} \lceil \mu_i e^{c_i t} \\
\frac{1}{c_i} \ln \frac{\mathbf{x}' \lceil \mu_i}{\mathbf{x} \lceil \mu_i} &= w(\mathbf{x} \lceil mode) - (\mathbf{y} \lceil x) \\
&= \frac{1}{c_i} \ln(1+h) - (\mathbf{y} \lceil x) \quad [\text{by replacing } w(\mathbf{x} \lceil mode)] \\
&= \frac{1}{c_i} \left[\ln(1+h) - \ln \frac{\mathbf{x} \lceil \mu_i}{\mathbf{x} \lceil \mu_{min}} \right] \quad [\text{from (24)}] \\
\mathbf{x}' \lceil \mu_i &= (1+h)(\mathbf{x} \lceil \mu_{min}) \\
&= (1+h)(\mathbf{x}' \lceil \mu_{min}) \quad [i \text{ not unique min } \Rightarrow \mu_{min} \text{ constant over } \tau'.]
\end{aligned}$$

Last equation implies that $\mathbf{x}'$ satisfies the stopping condition for $trajdef$ mode(i) for automaton $\mathcal{A}$. Therefore, $t = \tau.ltime = \tau'.ltime$. Thus we have shown that τ' is a valid trajectory of automaton $\mathcal{B}$.
We show that $\mathbf{x}' \mathcal{R} \mathbf{y}'$. First, $\mathbf{x}' \lceil mode = \zeta(\mathbf{y}' \lceil mode)$ because $\mathbf{x}' \lceil mode = \mathbf{x} \lceil mode = \zeta(\mathbf{y} \lceil mode) = \zeta(\mathbf{y}' \lceil mode)$. Secondly, for all k, $k \neq i$, $\mathbf{x} \lceil \mu_i = \mathbf{x}' \lceil \mu_i$ and $(\mathbf{y} \lceil mode)[1][k] = (\mathbf{y}' \lceil mode)[1][k]$. Finally, we show that $\mathbf{x}' \lceil \mu_i = (\mathbf{x}' \lceil \mu_{min}) e^{c_i (\mathbf{y}' \lceil x)}$ by reasoning as follows:

$$\begin{aligned}
\mathbf{x}' \lceil \mu_i &= (\mathbf{x} \lceil \mu_i) e^{c_i \tau.ltime} \\
&= (\mathbf{x} \lceil \mu_{min}) e^{c_i (\mathbf{y} \lceil x + \tau.ltime)} \\
&= (\mathbf{x}' \lceil \mu_{min}) e^{c_i (\mathbf{y} \lceil x + \tau'.ltime)}
\end{aligned}$$

(5) α is a closed trajectory τ of $\mathcal{A}$ with $(\tau \downarrow mode)(0) = i$ for some $i \in \mathcal{P}$, such that i is the unique minimum at $\tau.fstate$.
We choose β to be the trajectory τ' of $\mathcal{B}$ with $\tau'.dom = \tau.dom$ determined by the following rules. Let $\mathbf{x} = \tau.fstate$ and $\mathbf{y} = \tau'.fstate$.

$$\begin{gathered}
(\mathbf{y} \lceil mode)[1][i] = 1, (\mathbf{y} \lceil mode)[2][i] = 1 + h, \\
\forall k, k \neq i, (\mathbf{y} \lceil mode)[1][k] = (1+h), (\mathbf{y} \lceil mode)[2][k] = 1 \\
\forall\, t \in \tau'.dom, (\tau' \downarrow x)(t) = \tfrac{1}{c_i} \ln \tfrac{\mathbf{x} \lceil \mu_i}{\mathbf{x} \lceil \mu_{min}} + t
\end{gathered}$$

The rest of the proof for this case is similar to that for case 4.

The proof of the second part, that $\mathcal{R}$ is a switching simulation relation from $\mathcal{B}$ to $\mathcal{A}$, follows from the same steps as above. □