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

Average dwell time (ADT) properties characterize the rate at which a hybrid system performs mode switches. In this paper, we present a set of techniques for verifying ADT properties. The stability of a hybrid system $\mathcal{A}$ can be verified by combining these techniques with standard methods for checking stability of the individual modes of $\mathcal{A}$.

We introduce a new type of simulation relation for hybrid automata—*switching simulation*—for establishing that a given automaton $\mathcal{A}$ switches more rapidly than another automaton $\mathcal{B}$. We show that the question of whether a given hybrid automaton has ADT τ_a can be answered either by checking an invariant or by solving an optimization problem. For classes of hybrid automata for which invariants can be checked automatically, the invariant-based method yields an automatic method for verifying ADT; for automata that are outside this class, the invariant has to be checked using inductive techniques. The optimization-based method is automatic and is applicable to a restricted class of initialized hybrid automata. 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 and the invariant-based methods 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

A preliminary version of this paper was presented at the 2006 Workshop on Hybrid Systems: Computation and Control, under the title Verifying Average Dwell Time by Solving Optimization Problems.

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 DARPA/AFOSR MURI F49620-02-1-0325 grant and by NSF's CSR program (Embedded and Hybrid Systems area) under grant NSF-CNS-0614993.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 20YY ACM 0000-0000/20YY/0000-100001 $5.00

1. INTRODUCTION

Rapid growth in communication and microprocessor technologies is fueling the development of complex embedded devices which are being deployed to perform 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 or *modes*. 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 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 technique for analyzing stability relies on finding a *Common Lyapunov function*, whose derivative along the trajectories of all the modes must satisfy suitable inequalities. When such a function cannot be found 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. In the switched system model, the details of the discrete mechanisms of a hybrid automaton, namely, the guards and the reset maps, are neglected. Instead, an exogenous *switching signal* brings about the mode switches.

Assuming that the dynamics of the individual modes have certain properties, one can characterize the class of switching signals, based only on the rate of switches and not the particular sequence of switches, that guarantee stability of the entire system. For example, if the individual modes of the automaton are stable, then the notion of *dwell time* [Morse 1996] and the more general *average dwell time (ADT)* [Hespanha and Morse 1999] precisely characterize such restricted classes of switching signals that guarantee stability of the whole system. In order to apply this ADT-based stability verification technique to an hybrid automaton $\mathcal{A}$ one has to (a) find Lyapunov functions for the individual modes of $\mathcal{A}$, and (b) check that all executions of $\mathcal{A}$ satisfy the required ADT property. Roughly, $\mathcal{A}$ has ADT τ_a

if, in every execution of $\mathcal{A}$, the rate of mode switches is $\frac{1}{\tau_a}$ plus some constant burst rate. That is, over almost every τ_a interval $\mathcal{A}$ performs at most one switch. A large average dwell time means that $\mathcal{A}$ spends enough time in each mode, so as to dissipate the transient energy gained through mode switches. In this paper we assume that the 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 for a general class of hybrid automata. Thus, we provide a missing piece in the toolbox for analysis of stability of hybrid automata.

Application of ADT verification techniques are not limited to deducing stability of hybrid automata with stable modes. In [Zhai et al. 2000] ADT properties have been used to deduce stability conditions for systems with mixed stable and unstable state models. Input-to-state stability in the presence of inputs [Vu et al. 2006] and stochastic stability of randomly switched systems [Chatterjee and Liberzon 2006] can also be verified with the aid of ADT-like properties. In many settings, the question of whether a system have a certain ADT is natural and interesting independent of its connection to stability. For example, in queuing systems the ADT properties are used to characterize burstiness of traffic [Cruz 1991]. Our ADT verification techniques are likely to be valuable in these other contexts as well.

1.1 Contributions

In order to verify ADT, we have to model both the discrete and continuous mechanisms in a hybrid system; we use the *Structured Hybrid Automaton (SHA)* model derived from the *Hybrid Input/Output Automata (HIOA)* of [Lynch et al. 2003].

(i) We define what it means for a given SHA to switch "faster than" another SHA, and introduce a new kind of simulation relation for inductively proving this relationship between pairs of SHAs. This gives a sound method for abstraction that preserves ADT properties.

(ii) We present a method for verifying whether a given SHA has a certain ADT. This method relies on checking whether or not a transformed version of $\mathcal{A}$, satisfies a certain invariant property. Unlike ADT properties (which are properties of executions), an invariant property is a predicate on the states, and hence, can be checked directly using suitable invariant checking tools such as HyTech [Henzinger et al. 1997], PHAVer [Frehse 2005], and TAME [Archer 2001]. This method is applicable to a general class of hybrid automata; however, the invariant can be checked automatically only for restricted classes of automata. For hybrid automata that are not amenable to automatic invariant checking, we describe deductive techniques for proving invariants, which can be partially automated using mechanical theorem provers such as the TAME interface for PVS [Owre et al. 1996].

(iii) We present a second method for verifying ADT properties which relies on solving certain optimization problems. In order to check if $\mathcal{A}$ 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 know that no such counterexample exists, and that

$\mathcal{A}$ has ADT τ_a. For certain classes of *initialized* SHAs $\mathsf{OPT}(\tau_a)$ can indeed be solved using standard mathematical programming tools, and hence, this method yields an automatic ADT verification technique.

The two ADT verification methods complement each other as they can be combined to find the ADT of SHAs. By way of illustration, we verify the ADT properties of several typical hybrid systems using abstraction, simulation and the two verification methods.

1.2 Organization

In Section 2 the Structured Hybrid Automaton (SHA) model is introduced and compared to other existing hybrid system models; the linguistic conventions used throughout the paper for describing SHAs are presented; stability and ADT properties of SHAs are defined. Section 3 defines the "faster than" relation between SHAs and presents a simulation-based inductive technique for proving this relationship, and establishes the soundness of this technique. Section 4 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 5 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 6 concludes this paper with a summary of contributions and some remarks on directions for future research.

2. STRUCTURED HYBRID AUTOMATA

In this section, first, we introduce the *Structured Hybrid Automata (SHA)* model—an automaton model that is derived from HIOA [Lynch et al. 2003] and tailored for the results in this paper. In Section 2.5 we introduce the linguistic conventions used throughout the paper for describing SHAs. In Section 2.6 we define the different notions of stability and the role of ADT criterion in stability analysis.

2.1 Comparison of SHA with existing models

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 of 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. Although this adds some notational overhead, but it is a convenient feature for modeling systems whose discrete state consists of 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 SHA framework 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.

2.2 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 $J + t$ to be the set $\{t' + t \mid t' \in J\}$. In the SHA framework, a *variable* is used to specify state components. Each variable $x \in X$ is associated with a *type*, which is the set of values that x can assume. Each variable is also associated with a *dynamic type* which governs how the value of x could evolve over time. A variable x is *discrete* if its dynamic type is the set of piece-wise constant functions from T to the type of x. A variable x is *continuous* if it is not discrete and its dynamic type is the set of piece-wise continuous functions from T to the type of x. In applications, discrete variables are used to model software state such as counters and stacks, and continuous variables are used to model physical quantities such as temperature, position, and velocity. We refer the reader to Chapter 2 of [Mitra 2007] for definitions of dynamic types and for examples of discrete and continuous variables. We assume that all variables are either discrete or continuous. For a set of variables X, denote the set of discrete and continuous variables by X_d and X_c, respectively.

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 $\tau_1 \frown \tau_2 : \tau_1.dom \cup (\tau_2.dom + t) \rightarrow X$ 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 $x : \tau.dom \rightarrow type(x)$ to be $x(t) := (\tau \downarrow x)(t)$, for any $t \in \tau.dom$.

2.3 Definition of Structured Hybrid Automata

The Structured Hybrid Automaton (SHA) 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, and hence, unlike HIOAs, SHAs do not have input/output variables and do not distinguish among input, output, and internal actions. On the other hand, the trajectories of an SHA are specified 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: i(1) for all $\mathbf{x} \in val(X)$, there exists solution τ of the differential and algebraic equations in 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 $traj(X, F)$.

The above definition of state models can be naturally extended to include Differential and Algebraic Inequalities (DAIs). The formal definition (see, [Mitra 2007]) calls for the introduction of the notion of weak solutions of DAIs, which we omit in this paper. Henceforth state models are allowed to have differential and algebraic equations and inequalities.

Definition 2.2. A *Structured Hybrid Automaton* (SHA) $\mathcal{A} = (X, Q, \Theta, A, \mathcal{D}, P)$ consists of (1) a set X of *variables*, including a special discrete variable called *mode*, (2) a set $Q \subseteq val(X)$ of states and a nonempty set $\Theta \subseteq Q$ of *start states*, (3) a set A of transition labels or *actions*, (4) a set $\mathcal{D} \subseteq Q \times A \times Q$ of *discrete transitions*, and (5) an indexed family $P = \{F_i\}_{i \in \mathcal{P}}$ 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, Pre_a, is the set of states from which a transition labeled by action a is enabled. Formally, $Pre_a := \{\mathbf{x} \in Q \mid \exists \mathbf{x}', \mathbf{x} \xrightarrow{a} \mathbf{x}' \in \mathcal{D}\}$.

An initialized SHA is one in which every mode switching transition resets the values of the variables, nondeterministically, by choosing the values from a set that is independent of the pre-state. Formally, 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 switch 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 action a. An SHA is *linear* if the V-DAIs of all its state models are linear and the precondition and the initialization predicates (restricted to the set of continuous variables) are described by linear inequalities. A linear SHA is *rectangular*[1] if the differential equations in all the state models have constant right hand sides. Initialized SHAs are suitable for modeling periodic systems and systems with reset timers. Rectangular dynamics is suitable for modeling drifting clocks, motion under constant velocity, fluid-level under constant flow, etc.

In this paper, we make the following two assumptions about SHAs

(1) The collection of state models $\mathcal{P}$ is finite. If a discrete transition changes the values of the continuous variables then it is a mode switch.

(2) 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 second part of assumption (1) is without loss of generality because the set of state models $\mathcal{P}$, can be redefined, possibly by adding new elements, such that this condition is met. Assumption (2) is essential for the validity of the average dwell time theorem of Hespanha and Morse (Theorem 2.4) which gives a sufficient condition for stability based on average dwell time. Relaxing this assumption and verifying more general sufficient conditions for stability of SHAs is an avenue for future research.

2.4 Executions and invariants

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 $\tau_i.lstate \overset{a_{i+1}}{\rightarrow} \tau_{i+1}.fstate$. 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 set of executions of $\mathcal{A}$ is denoted by $\mathsf{Execs}_{\mathcal{A}}$. 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 closed prefix of α with $\alpha'.ltime = t$. If α is a closed execution, then this definition extends to $\alpha(t)$ for $t = \alpha.ltime$.

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. The set of reachable

[1]This definition of rectangular SHA is more general than the commonly used one, as for example in [Henzinger and Kopke 1996]. In the latter, for each action a, the precondition Pre_a and the reset map R_a are required to be rectangles.

states of $\mathcal{A}$ is denoted by $\mathsf{Reach}_{\mathcal{A}}$. An *invariant property* or simply an *invariant* of $\mathcal{A}$ is a condition on X that holds in all reachable states of $\mathcal{A}$.

2.5 Linguistic conventions

The standard circle-arrow diagrams used for specifying states and transitions of hybrid automata become a little cumbersome for automata with many modes and transitions. Instead, in this paper, we use the `HIOA` Language [Mitra 2007] for specifying SHAs. In what follows, we briefly describe the semantics of this language with the code of Figure 1 as an example; see Chapter 3 of [Mitra 2007] for a complete description of the language.

Variable names, their static and dynamic types, and initial values are defined in the **variables** section (lines 3–6). All non-real-valued variables are considered to be discrete; a real valued variable declared using the **discrete** keyword is discrete, if the keyword is omitted, the variable is continuous.

Action names are declared in the **actions** section and the corresponding transitions are defined in the **transitions** section. The predicate following the **pre** 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 20–26) defines a state model for each mode in terms of an *invariant condition*, a *stopping condition*, and a set of differential equations. For example, in the leaking mode, the continuous variables evolve according to the simple differential equation $d(x) = 1$ (line 25); 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. There is a subtle difference between the role of the invariant and stopping conditions: the former defines parts of the states space over which a state model can be active and the latter force discrete transitions to occur by stopping trajectories.

```
1  automaton Burner(D1, D2)
2    where D1, D2 ∈ ℝ≥0
3  variables
4    mode ∈ {normal, leaking},
5      initially normal
6    x ∈ R, initially x = 0
7
8  actions
9    leak, repair
10
11 transitions
12   leak
13     pre mode = normal ∧ x ≥ D1
14     effect mode ← leaking, x ← 0
15   repair
16     pre mode = leaking ∧ x = D2
17     effect mode ← normal, x ← 0
18
19 trajectories
20   trajdef normal
21     evolve d(x) = 1
22
23   trajdef leaking
24     invariant x ≤ D2
25     evolve d(x) = 1
26     stop when x = D2
```

Fig. 1: Leaking gas burner

2.6 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 $\mathbb{R}^n$ which we denote as $|\cdot|$. In defining the different types of stability properties for $\mathcal{A}$, we assume[2] that each state model $F_i \in P$ of $\mathcal{A}$ has the origin as its common equilibrium point, that is, $F_i(0) = \mathbf{0}$ 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 closed execution α of $\mathcal{A}$,

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

and we say that $\mathcal{A}$ is *stable*. An SHA $\mathcal{A}$ is *asymptotically stable* if it is stable and there exists δ_2 such that

$$|\alpha(0)| \leq \delta_2 \Rightarrow \alpha(t) \rightarrow \mathbf{0} \quad as \ \ t \rightarrow \infty \tag{2}$$

If the above condition holds for all δ_2 then $\mathcal{A}$ is *globally asymptotically stable*.

In the above definitions, the constants are quantified prior to the executions, and hence, these notions of stability are *uniform over all executions*. In this paper, we will employ the term "uniform" in the more conventional sense to describe uniformity with respect to the initial time of observation. Thus, *uniform stability* guarantees that the stability property in question holds not just for all executions, but for any suffix of the executions, that is, for all reachable execution fragments. An SHA $\mathcal{A}$ is *uniformly stable* in the sense of Lyapunov, if for every $\epsilon > 0$ there exists a constant $\delta_1 = \delta_1(\epsilon) > 0$, such that for any reachable closed execution fragment α, $|\alpha(0)| \leq \delta_1$ implies that $|\alpha(t)| \leq \epsilon$, for all 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 $\delta_2 > 0$, such that for every $\epsilon > 0$ there exists a $T > 0$, such that for any reachable closed execution fragment α,

$$|\alpha(0)| \leq \delta_2 \Rightarrow |\alpha(t)| \leq \epsilon, \quad \forall t \geq T \tag{3}$$

It is said to be *globally uniformly asymptotically stable* if the above holds for all δ_2, with $T = T(\delta_2, \epsilon)$.

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,

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

[2]These definitions can be easily generalized to the case where the state models have some other common equilibrium point.

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. Informally, 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.

Theorem 2.4. *Suppose there exist positive definite, continuously differentiable functions $\mathcal{V}_i : \mathbb{R}^n \to \mathbb{R}^n$, for each $i \in P$, such that we have two positive numbers λ_0 and μ, and two strictly increasing continuous functions β_1, β_2 such that:*

$$\beta_1(|\mathbf{x}_c|) \leq \mathcal{V}_i(\mathbf{x}_c) \leq \beta_2(|\mathbf{x}_c|), \quad \forall \mathbf{x}_c, \quad \forall i \in \mathcal{P}, \tag{5}$$

$$\frac{\partial \mathcal{V}_i}{\partial \mathbf{x}_c} f_i(\mathbf{x}_c) \leq -2\lambda_0 \mathcal{V}_i(\mathbf{x}_c), \quad \forall \mathbf{x}_c, \quad \forall i \in \mathcal{P}, \text{ and} \tag{6}$$

$$\mathcal{V}_i(\mathbf{x}'_c) \leq \mu \mathcal{V}_j(\mathbf{x}_c), \quad \forall \mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}', \text{ where } i = \mathbf{x}' \lceil \text{mode and } j = \mathbf{x} \lceil \text{mode}. \tag{7}$$

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

Its worth making a few remarks about this theorem. First of all, it is well-known that if the state model $F_i, i \in \mathcal{P}$, is globally asymptotically stable, then there exists a Lyapunov function $\mathcal{V}_i$ that satisfies (5) and $\frac{\partial \mathcal{V}_i}{\partial \mathbf{x}_c} f_i(\mathbf{x}_c) \leq -2\lambda_i V_i(\mathbf{x}_c)$, for appropriately chosen $\lambda_i > 0$. As the index set $\mathcal{P}$ is finite a λ_0 independent of i can be chosen such that for all $i \in \mathcal{P}$, Equation (6) holds. The third assumption, Equation (7), restricts the maximum increase in the value of the current Lyapunov functions over any mode switch, by a factor of μ.

In [Hespanha and Morse 1999] and [Liberzon 2003] this theorem is presented for the switched system model which differs from the more general SHA model in two ways: (a) In the switched system model, all variables are continuous except for the *mode* variable which determines the active state model. In SHA, there are both discrete and continuous variables. (b) The (discrete) transitions of a switched system correspond to the switching signal changing the value of *mode*; values of continuous variables remain unchanged over transitions. In SHAs, transitions *can* change the value of continuous variables. For example, a stopwatch is typically modeled as a continuous variable that is reset by discrete transitions. The proof of Theorem 2.4 still works for the SHA model because for this analysis, it suffices to consider only those discrete transitions of SHAs that are also mode switches. Assumption (1) guarantees that non-mode switching transitions do not change the value of the continuous variables. Secondly, resetting continuous variables change the value of the Lyapunov functions, but Equation (7) guarantees that the change is bounded by a factor of μ.

Proof sketch for Theorem 2.4. This proof is adapted from the proof of Theorem 3.2 of [Liberzon 2003] which constructs an exponentially decaying bound on the Lyapunov functions of each mode along any execution. Suppose α is any execution of $\mathcal{A}$. Let $T = \alpha.ltime$ and $t_1, \ldots, t_N$ be instants of mode switches in α. We will find an upper-bound on the value of $\mathcal{V}_{\alpha(T)\lceil mode}(\alpha(T))$, where $\alpha(t) \lceil \text{mode} \triangleq i, i \in I$ if

and only if $\alpha(t) \in Inv_i$. We define a function $W(t) \triangleq e^{2\lambda_0 t}\mathcal{V}_{\alpha(t)\lceil mode}(\alpha(t))$. Using the fact that W is non-increasing between mode switches and Equation 7 it can be shown that $W(t_{i+1}) \leq \mu W(t_i)$. Iterating this inequality $N(\alpha)$ times we get $W(T) \leq \mu^{N(\alpha)}W(0)$, that is

$$\begin{aligned} e^{2\lambda_0 T}\mathcal{V}_{\alpha(T)\lceil mode}(\alpha(T)) &\leq \mu^N(\alpha)\mathcal{V}_{\alpha(0)\lceil mode}(\alpha(0)), \\ \mathcal{V}_{\alpha(T)\lceil mode}(\alpha(T)) &\leq e^{-2\lambda_0 T + N(\alpha)\log\mu}\mathcal{V}_{\alpha(0)\lceil mode}(\alpha(0)) \end{aligned}$$

If α has average dwell time τ_a, then

$$\begin{aligned} \mathcal{V}_{\alpha(T)\lceil mode}(\alpha(T)) &\leq e^{-2\lambda_0 T + (N_0 + \frac{T}{\tau_a})\log\mu}\mathcal{V}_{\alpha(0)\lceil mode}(\alpha(0)) \\ &\leq e^{N_0 \log\mu} e^{(\frac{\log\mu}{\tau_a} - 2\lambda_0)T}\mathcal{V}_{\alpha(0)\lceil mode}(\alpha(0)). \end{aligned}$$

Now, if $\tau_a > \frac{\log\mu}{2\lambda}$ then $\mathcal{V}_{\alpha(T)\lceil mode}(\alpha(T))$ converges to 0 as $T \to 0$. Then from (5) it follows that $\mathcal{A}$ is globally asymptotically stable.

3. EQUIVALENCE WITH RESPECT TO ADT: SWITCHING SIMULATIONS

In order 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. We formalize this notion of equivalence as follows:

Definition 3.1. 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 say that $\mathcal{A}$ *switches slower* than $\mathcal{B}$ and write this as $\mathcal{A} \leq_{switch} \mathcal{B}$. If $\mathcal{B} \leq_{switch} \mathcal{A}$ and $\mathcal{A} \leq_{switch} \mathcal{B}$ then we say $\mathcal{A}$ and $\mathcal{B}$ are *ADT-equivalent*.

We propose an inductive method for proving ADT-equivalence. In concurrency theory, simulation relations (see, e.g., [Lynch and Vaandrager 1996]) have been widely used to prove that the set of visible behavior of one automaton is included in that of another. 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. Hence, we define a new kind of simulation relation that gives us an inductive technique for proving the $\leq_{switch}$ relationship between a pair of SHAs.

Definition 3.2. 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) *(Start condition)* 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) *(Transition condition)* 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) *(Trajectory condition)* 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 τ of a particular state

model $\mathcal{S}$, 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 3.3. *Let $\mathcal{A}$ 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 $S_{\tau_a}(\alpha) \leq S_{\tau_a}(\beta)$.*

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:

Case 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 Definition 3.2 ensures that there exists such a $\beta_0.fstate$ because $\alpha_0.fstate \in \Theta_{\mathcal{A}}$. We use Property 3 of Definition 3.2 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 Definition 3.2 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.

Case 2: α is a finite sequence ending with a closed trajectory. Similar to first case.

Case 3: α is a finite sequence ending with an open trajectory. Similar to first case except that the final open trajectory τ of α is constructed using a concatenation of infinitely many closed trajectories of $\mathcal{A}$ such that $\tau = \tau_0 \frown \tau_1 \frown \ldots$. Then, working recursively, we construct a sequence $\beta_0 \beta_1 \ldots$ of closed execution fragments of $\mathcal{B}$ such that for each i, $\tau_i.lstate \mathcal{R} \beta_i.lstate$, $\beta_i.lstate = \beta_{i+1}.fstate$, and $\beta_i.ltime \leq \tau_i.ltime$. This construction uses induction on i, using Property 3 of Definition 3.2 in the induction step. Now, let $\beta = \beta_0 \frown \beta_1 \frown \ldots$. Clearly, β is an execution fragment of $\mathcal{B}$ and $\tau.fstate \mathcal{R} \beta.fstate$ and $\beta.ltime \leq \tau.ltime$.

□

Theorem 3.4. *If $\mathcal{A}$ and $\mathcal{B}$ are SHAs and $\mathcal{R}$ is a switching simulation relation from $\mathcal{A}$ to $\mathcal{B}$, then $\mathcal{A} \leq_{switch} \mathcal{B}$.*

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 3.3 we know that there exists a β such that $S_{\tau_a}(\beta) \geq S_{\tau_a}(\alpha)$, from which the result follows. □

Corollary 3.5. *Let $\mathcal{A}$ and $\mathcal{B}$ be SHAs. Suppose $\mathcal{R}_1$ and $\mathcal{R}_2$ be a switching simulation relation from $\mathcal{A}$ to $\mathcal{B}$ and from $\mathcal{B}$ to $\mathcal{A}$, respectively. Then, $\mathcal{A}$ and $\mathcal{B}$ are ADT-equivalent.*

Switching simulation relations and Corollary 3.5 give us an inductive method for proving that *any* given pair of SHA are equivalent with respect to switching speed, that is, average dwell time.

4. VERIFYING ADT: INVARIANT APPROACH

In general, verifying that a SHA satisfies a property that is quantified over all the executions, such as the ADT property, is hard. Our first approach for verifying ADT relies on simple transformations defined by a small set of history variables, that convert the ADT property to an equivalent invariant. Thus, to prove ADT it suffices to prove an invariant, and the latter can be accomplished using existing tools and techniques. This approach is applicable to any SHA, although, the degree of automation depends on the dynamics of the SHA in question. SHAs for which automated invariant checking is not possible, appropriately strengthened version of the invariant can be proved inductively. Such proofs can be partially automated using theorem provers (see, e.g., [Mitra and Archer 2005]). In Sections 4.3 and 4.4 we use this method to verify ADT of a simple leaking gas-burner and a scale-independent hysteresis switch.

4.1 Transformations for ADT verification

In order 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 $q \in \mathbb{Z}$ and $y \in \mathbb{R}_{\geq 0}$.
(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 variable q is a counter that increments every time there is a mode switch of $\mathcal{A}$, and the variable y is a timer; every τ_a time q is decremented by 1 by triggering the 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 4.1. *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 4.2. *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 4.1 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 $\mathcal{A}$. Let α' be the corresponding

execution of $\mathcal{A}(\tau_a)$, that is, the restriction of α' to the variables and actions of $\mathcal{A}$ equals α. 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', that is, α is the restriction of α' to the variables and actions of $\mathcal{A}$. 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 $\mathbf{x}' \lceil q \leq N_0$. □

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 transformation is acceptable for asymptotic stability, but it does not guarantee uniform stability. For uniform stability we want all reachable execution fragments of $\mathcal{A}$ to satisfy Equation (4). Consider an execution α of $\mathcal{A}$ such that $\alpha = \alpha_0 \frown \alpha_1$, where $\alpha_0.ltime = t_1$, and $\alpha.ltime = t_2$. Let $N(\alpha_1)$ and $S_{\tau_a}(\alpha_1)$ denote the number of mode switches and the number of extra mode switches (w.r.t. τ_a) over the execution fragment α_1. For α_1 to satisfy (4), we require that

$$N(\alpha_1) \leq N_0 + \frac{t_2 - t_1}{\tau_a}, \text{ or } \quad S_{\tau_a}(\alpha_1) \leq N_0,$$

which is not guaranteed by the invariant $q \leq N_0$. This is because it is possible for q to become negative and then rapidly return to zero, all the time being less than N_0. For uniform stability we need to show that the total change in q between any two reachable states is bounded by N_0. So, 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 also restrict the variable q to have only non-negative values, to obtain uniform stability.

Theorem 4.3. *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 4.1 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 executions (fragments) 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 $\mathbf{x}' \lceil q - \mathbf{x}' \lceil q_{min} \leq N_0$. □

4.2 Proving invariants

Using theorem 4.2 (and 4.3) we can verify whether τ_a is an ADT of a SHA $\mathcal{A}$, by checking whether $q \leq N_0$ (and $q - q_{min} \leq N_0$) is an invariant for $\mathcal{A}(\tau_a)$. Recall that for an SHA $\mathcal{A}(\tau_a)$ with set of variables X and set of states $Q \subseteq val(X)$, a predicate $\mathcal{I}$ on X is an invariant if $\mathsf{Reach}_{\mathcal{A}(\tau_a)} \subseteq \mathcal{I}$. Here the set of states satisfying the predicate is also denoted by $\mathcal{I}$. Given $\mathcal{A}(\tau_a)$ and a predicate $\mathcal{I}$ on X, how do we verify that $\mathcal{I}$ is an invariant of $\mathcal{A}(\tau_a)$? If the set of reachable states of $\mathcal{A}(\tau_a)$, $\mathsf{Reach}_{\mathcal{A}(\tau_a)}$, is computable then we can check if $\mathsf{Reach}_{\mathcal{A}(\tau_a)} \subseteq \mathcal{I}$. It has been known since [Henzinger et al. 1995] that computing $\mathsf{Reach}_{\mathcal{A}(\tau_a)}$, is decidable if $\mathcal{A}$ belongs to a restricted class, such as the class of rectangular, initialized SHAs. In Section 4.3 we verify the ADT of a hybrid system which falls within this class. Several other classes of hybrid automata with linear, polynomial, and exponential state models have been identified (see, reachability related papers in [Tomlin and Greenstreet 2002; Maler and Pnueli 2003; Alur and Pappas 2004; Morari and Thiele 2005; Hespanha and Tiwari 2006]), for which the reachability problem is decidable. These decidable classes generally fall within, what is called the class of *order minimal* or *o-minimal* hybrid automata [Lafferriere et al. 1999]. The notion of o-minimality comes from mathematical logic. Informally, a theory of real numbers is said to be o-minimal if every definable subset of reals is a finite union of points and intervals. Then, a hybrid system is o-minimal, if its initial states, preconditions, reset maps, invariant sets, stopping conditions, and DAIs are all definable in the same o-minimal theory. Currently a practical algorithm for computing the reachable sets of general o-minimal hybrid automata remains unknown, however, in practice, software tools such as Phaver [Frehse 2005] and Rsolver [**?**] can be used to automatically check invariant properties of certain subclasses, and also for automata that are not o-minimal. Another alternative is to compute an overapproximation of $\mathsf{Reach}_{\mathcal{A}(\tau)}$ (see, for example, [Bayen et al. 2002]), say $\overline{\mathsf{Reach}_{\mathcal{A}(\tau_a)}}$, and then check if $\overline{\mathsf{Reach}_{\mathcal{A}(\tau_a)}} \subseteq \mathcal{I}$. If true then $\mathcal{I}$ is verified, otherwise one has to check if the states in $\overline{\mathsf{Reach}_{\mathcal{A}(\tau_a)}} \cap \mathcal{I}$ are really reachable or if they are false-positives generated by the overapproximation. In the latter case $\overline{\mathsf{Reach}_{\mathcal{A}(\tau_a)}}$ is iteratively refined.

For SHAs that do not admit any of the above automated techniques, invariants can be deduced partially automatically by: (a) finding an inductive property $\mathcal{I}' \subseteq \mathcal{I}$, and (b) checking that the transitions and trajectories of $\mathcal{A}$ preserve $\mathcal{I}'$. This technique has been applied extensively for verifying invariant properties of various hybrid systems arising in air-traffic control [Livadas et al. 1999; Umeno and Lynch

2007] and vehicle control [Mitra et al. 2003; Lynch 1996; Weinberg and Lynch 1996; Weinberg et al. 1995].

Definition 4.4. Let $\mathcal{A}$ be a SHA with set of internal variables X and set of states $Q \subseteq val(X)$. A predicate $\mathcal{I}$ on X is an *inductive property* of $\mathcal{A}$ if any execution that starts from a state satisfying $\mathcal{I}$ reaches only states that also satisfy $\mathcal{I}$.

An inductive property that is satisfied by all starting states of $\mathcal{A}$ is an invariant. Inductive properties can be verified by checking that each transition and trajectory of $\mathcal{A}$ preserves the property. The following lemma which is an adaptation of the classical inductive proof schema à la Floyd [Floyd 1967] to the SHA setting, formalizes this verification task.

Lemma 4.5. *Given a SHA $\mathcal{A}$, the set of states $\mathcal{I} \subseteq Q$ is an invariant of $\mathcal{A}$ if it satisfies:*

(1) (Start condition) For any starting state $\mathbf{x} \in \Theta$, $\mathbf{x} \in \mathcal{I}$.

(2) (Transition condition) For any action $a \in A$, if $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ and $\mathbf{x} \in \mathcal{I}$ then $\mathbf{x}' \in \mathcal{I}$.

(3) (Trajectory condition) For any state model $S \in \mathcal{P}$, any trajectory $\tau \in trajs(\mathcal{S})$, if $\tau.fstate \in \mathcal{I}$ then $\tau.lstate \in \mathcal{I}$.

PROOF. For any state $\mathbf{x} \in \mathsf{Reach}_{\mathcal{A}}$ there exists an execution α of $\mathcal{A}$ such that $\alpha.lstate = \mathbf{x}$. The proof is by induction on the length of the execution α. For the base case, α is consisting of a single starting state $\mathbf{x} \in \Theta$ and by the *start condition*, $\mathbf{x} \in \mathcal{I}$. For the inductive step, we consider two subcases:

Case 1: $\alpha = \alpha' a \mathbf{x}$ where a is an action of $\mathcal{A}$. By the induction hypothesis we know that $\alpha'.lstate \in \mathcal{I}$. Invoking the *transition condition* we obtain $\mathbf{x} \in \mathcal{I}$.

Case 2: $\alpha = \alpha' \tau$ where τ is a trajectory of $\mathcal{A}$ and $\tau.lstate = \mathbf{x}$. By the induction hypothesis, $\alpha'.lstate \in \mathcal{I}$. Since τ is a trajectory of $\mathcal{A}$, it follows that there exists a state model $\mathcal{S}$ such that $\tau \in trajs(\mathcal{S})$. Invoking the *trajectory condition* we get that $\tau.lstate = \mathbf{x} \in \mathcal{I}$.

□

In Section 4.4, we apply this deductive technique for verifying invariants, and hence ADT, of a noninitialized hybrid system.

4.3 Leaking gas burner

We illustrate the invariant-based ADT verification technique with the aid of the leaking gas-burner example from [Alur et al. 1993]. The gas-burner system (see, Figure 1) 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, $D_2 < D_1$. Mode switches are brought about by the leak and repair actions. Indeed, the ADT of this simple hybrid automaton is $\frac{D_1+D_2}{2}$. In this hybrid automaton model the individual modes are not stable and hence, ADT is verified only to illustrate the aforementioned technique, and it does *not* guarantee stability of the overall system.

In order 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 4.1.

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 Burner(τ_a). 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 Burner(τ_a). HyTech tells us that $q \leq N_0$ is an invariant for Burner(τ_a) only if $\tau_a \leq 12$, and not otherwise. It follows that the ADT of Burner with the above parameters is 12.

4.4 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:

$$\mu_i(0) \geq C_0 \tag{8}$$

$$\mu_{i^*}(t) \leq C_1 + C_2 e^{\lambda t}, \text{for some } i^* \in \{1, \ldots, m\} \tag{9}$$

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 4.2 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 4.1. The re-

sult is the automaton TRHSwitch of Figure 3. Lines 6–8, lines 27–29 and line 36 correspond to the transformation of Section 4.1. 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 $\perp$.

automaton 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)
  pre $mode = i \wedge (1+h)\mu_j \leq \mu_i$
  effect $mode \leftarrow j$

trajectories
  trajdef mode$_i$, $i \in \{1, \ldots, m\}$
  evolve $d(\mu) = f_i(\mu)$
  stop when $\exists \; j \in \{1, \ldots, m\}$, $(1 + h)\mu_j \leq \mu_i$

Fig. 2: Hysteresis switch

automaton 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
    $\mu_{min} = \min_i\{\mu_i\}$

  actions
    switch(i,j), i,$j \in \{1, \ldots, m\}$
    decrement

transitions
  switch(i,j)
    pre $mode = i \wedge (1+h)\mu_j \leq \mu_i$
    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 $\exists \; j \in \{1, \ldots, m\}$
    $(1 + h)\mu_j \leq \mu_i$
    $\vee \; y = (k + 1)\tau_a$

Fig. 3: Transformed 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 4.6. $q \leq c - \frac{y}{\tau_a} + 1$.

Invariant 4.7. *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 4.8. *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 $\mathbf{x} \lceil mode = j$ 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:

$$\mathbf{x}' \lceil \mu_i^{r+1} = \mathbf{x}' \lceil \mu_i = (1+h)(\mathbf{x}' \lceil \mu_j) \tag{10}$$

Let $\mathbf{x}''$ be the post state of the r^{th} $switch_i$ action. From the first part of Invariant 4.7, $(1+h)(\mathbf{x}'' \lceil \mu_j) \geq \mathbf{x}'' \lceil \mu_i = \mathbf{x}'' \lceil \mu_i^r$. 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 (10) 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 4.9. *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 4.6. 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 4.8 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 4.7 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 (8) of the monitoring signals, $\mu_j^1 \geq \mu_j^0 \geq C_0$. Therefore, for all $k \in \{1, \ldots, m\}$,

$$\begin{aligned}
\mathbf{x} \lceil c &\leq 1 + m + \frac{m}{\log(1+h)} \log\left(\frac{\mathbf{x} \lceil \mu_k}{C_0}\right). \\
&\leq 1 + m + \frac{m}{\log(1+h)} \log\left(\frac{C_1 + C_2 e^{\lambda\ (\mathbf{x} \lceil y)}}{C_0}\right), \quad \text{replacing } k \text{ with } i^* \text{ of (9)} \\
&\leq 1 + m + \frac{m}{\log(1+h)} \log\left(\frac{C_1 + C_2}{C_0}\right) + \frac{\lambda m\ (\mathbf{x} \lceil y)}{\log(1+h)}
\end{aligned}$$

Using Invariant 4.6, and putting $\tau_a = \frac{\log(1+h)}{\lambda m}$, we get the result. □

From the above invariant and Theorem 4.2 it is established that HSwitch has an average dwell time of at least $\frac{\log(1+h)}{\lambda m}$. The following property of the switch is a consequence of the ADT property and the assumptions on the monitoring signals, and it states the desirable property in terms of switches between controllers.

Theorem 4.10. *If there exists an index $i \in \mathcal{P}$ such that the monitoring signal μ_i is bounded then the the switching between controllers stop in finite time at some index $j \in I$ and μ_j is bounded.*

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.

5. OPTIMIZATION BASED APPROACH

In this section we develop the second method for verifying ADT properties. 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 SHAs, the resulting optimization problem can be solved efficiently. This approach is complete for sub-classes of initialized SHAs and yields an automatic method for verifying ADT. The hardness of the resulting optimization problem depends on the dynamics of the SHA in question.

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:

$$\mathsf{OPT}(\tau_a): \qquad \max S_{\tau_a}(\alpha)$$

where α is an 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.

5.1 One-clock initialized SHA

Recall the definition of initialized SHAs from Section 2.3. Here we consider a special class of initialized SHA, called *one-clock initialized SHA*. As the name suggests, such automata have a single clock variable which is reset at every mode switch. Clearly, computing reachable states is decidable for one-clock initialized SHAs and therefore,

we can apply the invariant based technique of Section 4 and model-checking to automatically verify ADT properties. Yet we apply the optimization-based ADT verification technique here because, as we shall see shortly, $\mathsf{OPT}(\tau_a)$ can be solved efficiently using classical graph algorithms for this class. In Section 5.3, we consider the case of general initialized SHAs.

A weighted directed graph uniquely defines a one-clock initialized SHA. Consider a directed graph $G = (\mathcal{V}, \mathcal{E}, w, e_0)$, where (1) $\mathcal{V}$ is a finite set of vertices, (2) $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of directed edges, (3) $w : \mathcal{E} \to \mathbb{R}_{\geq 0}$ is a cost function for the edges, and (4) $e_0 \in \mathcal{E}$ is a special *start edge*. The cost of a path in G is the sum of the costs of the edges in the path. Given the graph G, 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.

---

automaton $Aut(G)$ **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
  $\mathsf{switch}(e, e')$, $e, e' \in \mathcal{E}$

transitions
  $\mathsf{switch}(e, e')$
  pre $mode = e \wedge e[2] = e'[1] \wedge x = w(e)$
  effect $mode \leftarrow e'$, $x \leftarrow 0$

trajectories
  trajdef $\mathsf{edge}(mode)$
  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 5.1. *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.

For the other direction, 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,

$$l_1 + l_3 > N_0 + \frac{1}{\tau_a}\left[\sum_{i=1}^{l_1} w_i + \sum_{i=1}^{l_3} w_i\right] \tag{11}$$

From Equation (11), $S_{\tau_a}(\beta_p \frown \beta_s) > N_0$, 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].

5.2 Linear hysteresis switch

Using Theorem 5.1 and switching simulations we verify the ADT of a linear version of the HSwitch automaton of Figure 2 in Section 4.4. 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.

---

automaton 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
- $\mu_{min} = \min_i\{\mu_i\}$

actions

transitions

switch(i,j)
- **pre** $mode = i \wedge (1+h)\mu_j \leq \mu_i$
- **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** $\exists\ j \in \{1, \ldots, m\}\ (1+h)\mu_j \leq \mu_i$

Fig. 5: Linear hysteresis switch

The LinHSwitch automaton is not a one-clock initialized SHA. We cannot apply Theorem 5.1 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 $(mode=1,[\mu_1=(1+h)C_0,\mu_2=C_0,\mu_3=C_0])$, $a_1=\mathsf{switch}(1,3)$, $\tau_1.dom=[0,\frac{1}{c_3}\ln(1+h)]$, $(\tau_1\downarrow\mu_3)(t)=C_0e^{c_3t}$, $a_2=\mathsf{switch}(3,2)$, $\tau_2.dom=[0,\frac{2}{c_2}\ln(1+h)]$, $(\tau_2\downarrow\mu_2)(t)=C_0e^{c_2t}$. 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.

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 5.2. 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 5.2 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 $j=\zeta(e)$, $\mu_j=\mu_{min}e^{c_jx}$.

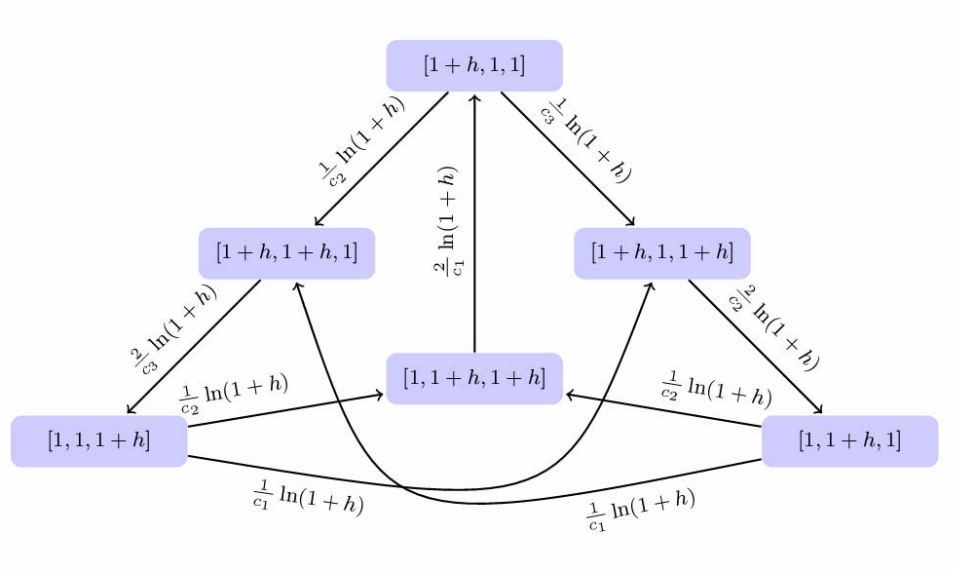

Fig. 6: ADT-equivalent graph ($m = 3$) for LinHSwitch.

Lemma 7.1 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 complete proof is given in Appendix A.

Lemma 5.3. *$\mathcal{R}$ is a switching simulation relation from $\mathcal{A}$ to $\mathcal{B}$.*

From Corollary 3.5 it follows that LinHSwitch $\leq_{switch} Aut(G)$, and hence, if τ_a is an ADT for $Aut(G)$ then it is also an ADT for LinHSwitch. As $Aut(G)$ is one-clock initialized SHA, from the results in Section 5.1, we know that we can verify whether τ_a is an ADT of $Aut(G)$ efficiently by finding the minimum mean cost cycle of G. If it is, we can conclude that ADT of LinHSwitch is also at least τ_a.

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 4.9 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 4.9 is based on upper and lower bounds given by Equations (8) and (9).

5.3 Initialized SHA

In this section we study ADT properties of initialized SHA. Recall that every action $a \in A$ of an initialized SHA $\mathcal{A}$ is associated with two sets of states, an initialization set R_a and a precondition Pre_a, such that $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ is a (mode switching) discrete transition if and only if $\mathbf{x} \in Pre_a$ and $\mathbf{x}' \in R_a$.

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 5.4. *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 $\alpha = \tau_0 a_1 \tau_1 \ldots \tau_n$ be a closed execution that has the smallest number of mode switches. From α, we construct $\beta = \tau_0^* a_1 \tau_1^* \ldots \tau_n^*$, 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 $i, j \in \{1, \ldots, m\}$, such that $a_i = a_j$ and $a_{i+1} = a_{j+1}$, then we make $\tau_i^* = \tau_j^*$.

Claim 5.5. *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 $i, j \in \{1, \ldots, m\}$, 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 5.6. *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 5.4. □

Theorem 5.7. *Suppose $\mathcal{A}$ is an initialized SHA with state models indexed by $\mathcal{P}$. For any $\tau_a > 0$, τ_a is an ADT for $\mathcal{A}$ if an only if all cycles of length at most $|\mathcal{P}|$ are free of extra switches.*

Proof. Follows from Corollary 5.6 and the definition of the optimization problem $\mathsf{OPT}(\tau_a)$. □

Theorem 5.7 gives us a method for verifying ADT of initialized SHAs by maximizing $\mathsf{OPT}(\tau_a)$ over all cycles of length at most $|\mathcal{P}|$. In other words, for verify ADT of initialized hybrid systems it suffices to solve the optimization problem over a much smaller set of executions than we set out with at the beginning of Section 5. For non-initialized SHA $\mathcal{A}$, the first part of Theorem 5.4 holds. That is, solving $\mathsf{OPT}(\tau_a)$ over all cycles of length at most $|\mathcal{P}|$, if a cycle with extra switches is found, it follows that τ_a is not an ADT for $\mathcal{A}$. Solving $\mathsf{OPT}(\tau_a)$ relies on formulating it as a mathematical program such that standard mathematical programming tools can be used. This is the topic of the next section.

It is worth noting that this method can be combined with the invariant-based method of Section 4 for finding the ADT of a given SHA. We 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.

5.4 MILP formulation of OPT(τ_a)

An SHA is *rectangular* if the differential equations in the state models have constant right hand sides. In this section, we assume in addition that that precondition and the initialization predicates (restricted to the set of continuous variables) are polyhedral. We show how the $\mathsf{OPT}(\tau_a)$ can be formulated as a Mixed Integer Linear Program (MILP) for verifying ADTs of rectangular initialized SHAs. MILP-based techniques have been previously used to verify safety properties of hybrid systems (see, for example, [Bemporad and Morari 1999]).

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 $\mathsf{switch}(i, j)$. 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.

---

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

transitions
$\mathsf{switch}(p,q)$
pre $mode = p \wedge G[p,q]\mathbf{x} \leq g[p,q]$
effect $mode \leftarrow q$
$\mathbf{x} \leftarrow \mathbf{x}'$ such that $R[p,q]\mathbf{x}' \leq r[p,q]$

trajectories
trajdef $\mathsf{mode}(p)$
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 $\mathsf{MOPT}(K, \tau_a)$ 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 5.6 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 $\mathsf{MOPT}(K, \tau_a)$'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

$$m_{uj} = \begin{cases} 1, & \text{if mode over } u^{th} \text{ trajectory is } j \\ 0, & \text{otherwise.} \end{cases} \quad \text{for each } u \in \{0, 2, \ldots, K\},\ j \in \{1, \ldots, N\}$$

$$p_{ujk} = \begin{cases} 1, & \text{if mode over } (u-1)^{st} \text{ trajectory is } j \text{ and over } (u+1)^{st} \text{ trajectory is } k \\ 0, & \text{otherwise.} \end{cases} \quad \text{for each } u \in \{0, 2, 4, \ldots, K\}, j, k \in \{1, \ldots, N\}$$

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 (12) in Figure 8. For each odd u, Constraints (14) and (15) ensure that $(\mathbf{x}_u, \mathsf{switch}(j, k), \mathbf{x}_{u+1})$

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

Mode: $\forall\ u \in \{0, 2, \ldots, K\},\ \displaystyle\sum_{j=1}^{N} m_{uj} = 1$ *and* $\forall\ u \in \{1, 3, \ldots, K-1\},\ \displaystyle\sum_{j=1}^{N}\sum_{k=1}^{N} p_{ujk} = 1$ (12)

Cycle: $\mathbf{x}_0 = \mathbf{x}_K$ and $\forall\ j \in \{1, \ldots, N\}, m_{0j} = m_{Kj}$ (13)

Preconds: $\forall\ u \in \{1, 3, \ldots, K-1\},\ \displaystyle\sum_{j=1}^{N}\sum_{k=1}^{N} G[j,k].p_{ujk}.\mathbf{x}_u \leq \sum_{j=1}^{N}\sum_{k=1}^{N} p_{ujk}.g[j,k]$ (14)

Initialize: $\forall\ u \in \{1, 3, \ldots, K-1\},\ \displaystyle\sum_{j=1}^{N}\sum_{k=1}^{N} R[j,k].p_{ujk}.\mathbf{x}_{u+1} \leq \sum_{j=1}^{N}\sum_{k=1}^{N} p_{ujk}.r[j,k]$ (15)

Invariants: $\forall\ u \in \{0, 2, \ldots, K\},\ \displaystyle\sum_{j=1}^{N} A[j].m_{uj}.\mathbf{x}_u \leq \sum_{j=1}^{N} m_{uj}.a[j]$ (16)

Evolve: $\forall\ u \in \{0, 2 \ldots, K\},\ \mathbf{x}_{u+1} = \mathbf{x}_u + \displaystyle\sum_{j=1}^{N} c[j].m_{uj}.t_u$ (17)

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 (16) 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 (16). 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 (17) 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 (16) 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 $\mathbf{x}_u$. This linearization increases the size of the MILP that we have to solve but does not affect the soundness or the completeness of the verification method.

5.5 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, SHA Thermostat (see Figure 9 *Left*), 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 $\leq_{switch}$ 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 5.8. For any $\mathbf{x} \in Q_{\mathsf{Thermostat}}$ and $\mathbf{y} \in Q_{\mathsf{Approx}}$, $\mathbf{x}\ \mathcal{R}\ \mathbf{y}$ 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 5.9. *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 similar to that of Lemma 7.1. 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 5.9 implies that Thermostat $\leq_{switch}$ 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 $\tau_a =$

automaton 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
pre $mode = l_0 \wedge x \leq \theta_2$
effect $mode \leftarrow l_1$

off
pre $mode = l_1 \wedge x \geq \theta_3$
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 $\mathbf{d(x) = C(h\text{-}x)}$; $\mathbf{d(z) = 1}$
invariant $x \leq \theta_4$
stop when $x = \theta_4$

automaton 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$_1$
pre $mode = l_0 \wedge r \geq L_0$
effect $mode \leftarrow l_1$, $r \leftarrow 0$

switchto$_0$
pre $mode = l_1 \wedge r \geq L_1$
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.

$25, 27, 28$, 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 $\leq_{ADT}$ Approx, we conclude that the ADT of the thermostat is no less than 27.

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 5.4 does not hold and therefore from the failure to find a counterexample we cannot conclude that the automaton satisfies the ADT property in question.

6. CONCLUSIONS

In this paper we have presented two methods for proving ADT properties of hybrid systems. Stability of a hybrid system is guaranteed if 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 is applicable to any hybrid system; however, automatic verification is possible only for those classes of systems for which invariant properties can be checked automatically. For hybrid systems outside these classes,

semi-automatic inductive 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 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 and proposed a new kind of simulation relation, namely switching simulation, which gives a sufficient condition for establishing this equivalence relationship.

There are several research directions to be pursued related to stability verification of SHAs. One interesting problem is to develop stability verification techniques for the general class of SHAs that have both stable and unstable state models. Sufficient conditions for stability of such systems already exist in the control theory literature (see, for example [Zhai et al. 2000]). These conditions take the form of switching time related properties and are hard to verify, just like the ADT property, and hence, they call for the development of new verification techniques. Another direction, is to include input/output variables in the automaton model and explore verification of input-output and input-to-state stability properties using the results from [Vu et al. 2006]. Yet another direction of future research is to extend these techniques to stochastic hybrid systems, by combining the probabilistic timed I/O automata of [Mitra and Lynch 2007] with ADT-like stability results for stochastic switched systems from [Chatterjee and Liberzon 2006].

Acknowledgments

We thank Debasish Chatterjee for giving us many useful comments and suggestions on this paper. We thank Andy Teel for suggesting an alternative to our formulation of the invariant for uniform stability.

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.

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. LNCS, vol. 736. Springer-Verlag, 209–229.

Alur, R. and Pappas, G. J., Eds. 2004. *Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, Proceedings.* LNCS, vol. 2993. Springer.

Archer, M. 2001. TAME: PVS Strategies for special purpose theorem proving. *Annals of Mathematics and Artificial Intelligence 29,* 1/4 (February).

Bayen, A. M., Cruck, E., and Tomlin, C. 2002. Guaranteed overapproximations of unsafe sets for continuous and hybrid systems: solving the hamilton-jacobi equation using viability techniques. See Tomlin and Greenstreet [2002], 90–104.

Bemporad, A., Bicchi, A., and Buttazzo, G. C., Eds. 2007. *Hybrid Systems: Computation and Control, 10th International Workshop, HSCC 2007, Pisa, Italy, April 3-5, 2007, Proceedings*. LNCS, vol. 4416. Springer.

Bemporad, A. and Morari, M. 1999. Verification of Hybrid Systems via Mathematical Programming. *Hybrid Systems: Computation and Control 1569*, 31–45.

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

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

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.

Chatterjee, D. and Liberzon, D. 2006. Stability analysis of deterministic and stochastic switched systems via a comparison principle and multiple lyapunov functions. *SIAM Journal on Control and Optimization 45,* 1 (March), 174–206.

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

Cruz, R. L. 1991. A calculus for network delay, part i: Network elements in isolation. *IEEE Transactions on Information Theory 37,* 1, 114–131.

Floyd, R. 1967. Assigning meanings to programs. In *Sympyposium on Applied Mathematics. Mathematical Aspects of Computer Science*. American Mathematical Society, 19–32.

Frehse, G. 2005. Phaver: Algorithmic verification of hybrid systems past hytech. See Morari and Thiele [2005], 258–273.

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

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.

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

Henzinger, T. A. and Kopke, P. W. 1996. State equivalences for rectangular hybrid automata. In *International Conference on Concurrency Theory (CONCUR'96)*. 530–545.

Henzinger, T. A., Kopke, P. W., Puri, A., and Varaiya, P. 1995. What's decidable about hybrid automata? In *ACM Symposium on Theory of Computing*. 373–382.

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)*. LNCS 1785, Springer-Verlag, 142–156.

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

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.

Hespanha, J. P. and Tiwari, A., Eds. 2006. *Hybrid Systems: Computation and Control, 9th International Workshop, HSCC 2006, Santa Barbara, CA, USA, March 29-31, 2006, Proceedings*. LNCS, vol. 3927. Springer.

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

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

Lafferriere, G., Pappas, G. J., and Yovine, S. 1999. A new class of decidable hybrid systems. In *HSCC '99: Proceedings of the Second International Workshop on Hybrid Systems*. Springer-Verlag, London, UK, 137–151.

Liberzon, D. 2003. *Switching in Systems and Control*. Systems and Control: Foundations and Applications. Birkhauser, Boston.

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.

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.

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

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

Maler, O. and Pnueli, A., Eds. 2003. *Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3-5, 2003, Proceedings*. LNCS, vol. 2623. Springer.

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

Mitra, S. 2007. A verification framework for hybrid systems. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA 02139.

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

Mitra, S. and Lynch, N. A. 2007. Trace-based semantics for probabilistic timed i/o automata. See Bemporad et al. [2007], 718–722. Full version http://theory.lcs.mit.edu/~mitras/research/PTIOA06-full.pdf.

Mitra, S., Wang, Y., Lynch, N., and Feron, E. 2003. Safety verification of model helicopter controller using hybrid Input/Output automata. See Maler and Pnueli [2003], 343–358.

Morari, M. and Thiele, L., Eds. 2005. *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings*. LNCS, vol. 3414. Springer.

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.

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 LNCS. Springer-Verlag, New Brunswick, NJ, 411–414.

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

Tomlin, C. and Greenstreet, M. R., Eds. 2002. *Hybrid Systems: Computation and Control, 5th International Workshop, HSCC 2002, Stanford, CA, USA, March 25-27, 2002, Proceedings*. LNCS, vol. 2289. Springer.

Umeno, S. and Lynch, N. A. 2007. Safety verification of an aircraft landing protocol: A refinement approach. See Bemporad et al. [2007], 557–572.

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

Vu, L., Chatterjee, D., and Liberzon, D. 2006. Input-to-state stability of switched systems and switching adaptive control. *Automatica*.

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.

Weinberg, H. B., Lynch, N., and Delisle, N. 1995. Verification of automated vehicle protection systems. In *Hybrid Systems III: Verification and Control Workshop on Verification and Control of Hybrid Systems)*, T. H. R. Alur and E. Sontag, Eds. LNCS, vol. 1066. Springer-Verlag, 101–113.

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

Zhai, G., Hu, B., Yasuda, K., and Michel, A. 2000. Stability analysis of switched systems with stable and unstable subsystems: An average dwell time approach. In *Proceedings of the 2000 American Control Conference (2000)*. chicago,illinois.

7. APPENDIX A

Lemma 7.1. *$\mathcal{R}$ 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:

$$e[1][i] = 1, e[2][i] = 1 + h, \forall k, k \neq i, e[1][k] = e[2][k] = \frac{\mathbf{x} \lceil \mu_k}{\mathbf{x} \lceil \mu_{min}} \tag{18}$$

$$e'[1][j] = 1, e'[2][j] = 1 + h, \forall k, k \neq j, e'[1][k] = e'[2][k] = \frac{\mathbf{x}' \lceil \mu_k}{\mathbf{x}' \lceil \mu_{min}} \tag{19}$$

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:

$$(\mathbf{y} \lceil mode)[1][i] = 1, (\mathbf{y} \lceil mode)[2][i] = i + h,$$
$$\forall k, k \neq i, (\mathbf{y} \lceil mode)[1][k] = (\mathbf{y} \lceil mode)[2][k] = \frac{\mathbf{x} \lceil \mu_k}{\mathbf{x} \lceil \mu_{min}} \tag{20}$$

Comparing Equations (20) and (18) we conclude that $\mathbf{y} \lceil mode = e$.
Secondly, using the definitions of e, e' and $\mathcal{R}$ it follows that:

$$e[2][i] = 1 + h = \frac{\mathbf{x} \lceil \mu_i}{\mathbf{x} \lceil \mu_{min}} = \frac{\mathbf{x}' \lceil \mu_i}{\mathbf{x}' \lceil \mu_{min}} = e'[1][i] \tag{21}$$

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:

$$e[2][j] = \frac{\mathbf{x} \lceil \mu_j}{\mathbf{x} \lceil \mu_{min}} = \frac{\mathbf{x}' \lceil \mu_j}{\mathbf{x}' \lceil \mu_{min}} = 1 = e'[1][j] \tag{22}$$

$$\forall k, k \neq j, k \neq i, e[2][k] = \frac{\mathbf{x} \lceil \mu_k}{\mathbf{x} \lceil \mu_{min}} = \frac{\mathbf{x}' \lceil \mu_k}{\mathbf{x}' \lceil \mu_{min}} = e'[1][k] \tag{23}$$

Combining Equations (21),(21) and (23) 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 (19). 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 (19) it follows that $\frac{\mathbf{x}' \lceil \mu_k}{\mathbf{x}' \lceil \mu_{min}} = e'[1][k] = e'[2][k]$.

(2) 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 $\mathbf{y}' = \tau'.lstate$.

$$\begin{aligned} (\mathbf{y} \lceil mode)[1][i] = 1, (\mathbf{y} \lceil mode)[2][i] = 1 + h, \\ \forall k, k \neq i, (\mathbf{y} \lceil mode)[1][k] = (\mathbf{y} \lceil mode)[1][k] = \tfrac{\mathbf{x} \lceil \mu_i}{\mathbf{x} \lceil \mu_{min}}, \\ \forall\, t \in \tau'.dom, (\tau' \downarrow x)(t) = \tfrac{1}{c_i} \ln \tfrac{\mathbf{x} \lceil \mu_i}{\mathbf{x} \lceil \mu_{min}} + t \end{aligned} \tag{24}$$

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 $trajdef$ $\mathsf{mode(i)}$ 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:

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

The proofs for the remaining cases are similar to those presented above.

□