Abstraction of Dynamical Systems by Timed Automata

Wisniewski, Rafal; Sloth, Christoffer

Published in:
Modeling, Identification and Control (Online Edition)

DOI (link to publication from Publisher):
10.4173/mic.2011.2.3

Publication date:
2011

Document Version
Early version, also known as pre-print

Modeling, Identification and Control, Vol. 31, No. 1, 2010, pp. 1–12, ISSN 1890–1328

Abstraction of Dynamical Systems by Timed Automata

Rafael Wisniewski [1] Christoffer Sloth [2]

[1] *Section of Automation & Control, Aalborg University, Denmark. E-mail:* raf@es.aau.dk

[2] *Department of Computer Science, Aalborg University, Denmark. E-mail:* csloth@cs.aau.dk

Abstract

To enable formal verification of a dynamical system, given by a set of differential equations, it is abstracted by a finite state model. This allows for application of methods for model checking. Consequently, it opens the possibility of carrying out the verification of reachability and timing requirements, which by classical control methods is impossible. We put forward a method for abstracting dynamical systems, where level sets of Lyapunov functions are used to generate the partitioning of the state space. We propose to partition the state space using an entire family of functions. The properties of these functions ensure that the discrete model captures the behaviors of a dynamical system by generating appropriate equivalence classes of the states. These equivalence classes make up the partition of the state space.

Keywords: verification, stability, abstraction, timed automata

1 Introduction

Verifying that a dynamical system satisfies a specification is a complicated, but inevitable part of designing a system. Frequently, it is not possible to conduct the verification by simulation, as an exhaustive simulation of all initial conditions, disturbances, etc., is not possible. However, formal verification methods can be applied to dynamical system if a finite combinatorial abstraction can be devised. Apart from the automatization of the verification process, formal verification methods provide answers to entirely new type of questions in control engineering. A few examples of these are: Do all solutions of the dynamical system, initialized in a subset X_0 of the state space, reach the set of goal states X_{goal}? Do all solutions of the dynamical system, initialized in X_0, reach X_{goal} within 5 s? Does there exist a solution of the dynamical system, initialized in X_0 that passes the unsafe states X_{unsafe}?

In particular, the verification of system properties such as safety is based on reachability calculation or its approximation. The exact reachable sets of continuous and hybrid systems are in general incomputable Asarin et al. (2006). Therefore, much research effort has been made on the approximation of reachable sets, specially, for continuous systems Guéguen et al. (2009). Yet, reachability is decidable for system models given by automata and timed automata; consequently, there exists a rich set of tools aimed at verifying properties of such systems, e.g., UPPAAL, see Behrmann et al. (2004).

There are essentially two methods for verifying dynamical and hybrid systems Guéguen et al. (2009). The first method is to over-approximate the reachable states of a system by sets such as ellipsoids, cubes, and simplexes. This is accomplished in Kurzhanski and Vályi (1997) using ellipsoids, in Girard (2005) using zonotopes, and in Mitchell et al. (2005) using level sets of the Hamilton-Jacobi-Isaacs partial differential equation. The other method is to abstract a system by a model of reduced complexity, which preserves crucial dynamical properties of the original systems. This is accomplished in Maler and Batt (2008) for continuous

doi:10.4173/mic.2010.1.1

systems, and in Tiwari (2008) for hybrid systems. Both methods rely on explicit calculation of reachable sets of the continuous dynamics, which is the main source of complexity in the verification procedure Guéguen et al. (2009).

In this paper, we provide a method for abstracting dynamical systems by timed automata. This method is based on partitioning the state space using level sets of Lyapunov functions; hence, the partitioning is conducted according to the dynamics of the system as in Tiwari (2008); Prajna (2006). This allows the resulting combinatorial model to be relatively small. Contrary to Tiwari (2008); Prajna (2006), we generate a timed model, which enables the verification of timed temporal-logic specifications Alur et al. (1990). Additionally, in contrast to Frehse (2005), the proposed abstraction-procedure does not use solutions to the system equations or any kind of simulation. Furthermore, the proposed method permits a parallel composition, which is vital for the verification of high-dimensional systems.

This paper is organized as follows. Section 2 recalls definitions from dynamical systems. The notion of timed automaton, the definition, and qualities are presented in Section 3. Section 4 presents the proposed abstraction procedure along with its properties. An example in Section 5 demonstrates this procedure. Finally, Section 6 comprises conclusions.

2 Dynamical Systems

Let $\mathbb{E}$ denote the Euclidean space $\mathbb{R}^n$ with the standard scalar product $\langle x, y\rangle = x^{\mathrm{T}}y$. Occasionally, we will indicate the dimension of $\mathbb{E}$ by writing $\mathbb{E}^n$. We address the verification problem of an autonomous dynamical system $\Gamma = (X, \xi)$, where $X \subset \mathbb{E}$ is a state space, which will be specified later in this article, and $\xi : \mathbb{E} \to \mathbb{E}$ is a continuous locally Lipschitz vector field. We denote the set of critical point of ξ by $\mathrm{Cr}(\xi) \equiv \{x \in \mathbb{E} |\ \xi(x) = 0\}$. For an ordinary differential equation

$$\dot{x} = \xi(x), \tag{1}$$

a flow map $\Phi_\Gamma : [0, \epsilon] \times \mathbb{E} \to \mathbb{E}$ for an $\epsilon > 0$ satisfies

$$\frac{d\Phi_\Gamma(t, x)}{dt} = \xi\left(\Phi_\Gamma(t, x)\right) \tag{2}$$

for all $t \in [0, \epsilon]$. In other words, $\Phi_\Gamma(t, x)$ is the solution of (1) from an initial state x in a set of initial states $X_0 \subseteq X$ and for time $t \in [0, \epsilon]$.

Given a system $\Gamma = (X, \xi)$, a set $R \subseteq X$ is said to be positively invariant if for all $x \in R$ and for all $t \in \mathbb{R}_+$ (the set of nonnegative reals)

$$\Phi_\Gamma(t, x) \in R. \tag{3}$$

In particular, the above definition implies that the Φ_Γ is defined for all $x \in R$ and for all nonnegative time t. In this article, we will often use the following notation. For a map $f : A \to B$, and a subset $C \subseteq A$, $f(C) \equiv \{f(x) |\ x \in C\}$. Thus in particular, R is positively invariant if $\Phi_\Gamma(\mathbb{R}_+, R) \subseteq R$.

Invariant sets play a crucial role in this work. Indeed, we will use the observation that, if the set of initial conditions and the goal-sets are subsets of a positively invariant set, and the unsafe states are in its complement, then the system is safe. That is, there are no trajectories that enter the unsafe-set. This situation is illustrated in Figure 1.

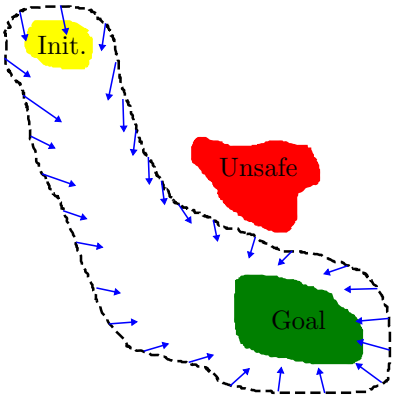

Figure 1: The set of initial states and the goal-set are subsets of a positive invariant set; whereas, the unsafe states are in its complement.

Besides invariant, also reachable sets are instrumental for verification.

Definition 1 (Reachable set of Dyn. System) *The reachable states of a system Γ from a set of initial states $X_0 \subseteq X$ on the time interval $[t_1, t_2]$ is defined as*

$$\Phi_\Gamma([t_1, t_2], X_0). \tag{4}$$

Thus, the reachable set of X_0 on time interval $[t_1, t_2]$ consists of all those points p for which there exists a flow line (a trajectory) connecting the set X_0 with p, and it takes time t in the interval $[t_1, t_2]$.

3 Timed Automata

In the sequel, we will abstract dynamical systems by timed automata. A timed automaton consists of discrete locations, transitions between locations, which are labeled by actions and clocks which may be reset to zero whenever a transition takes place. A timed automaton is illustrated in Figure 2. The locations are denoted by p and q, where the initial location is p;

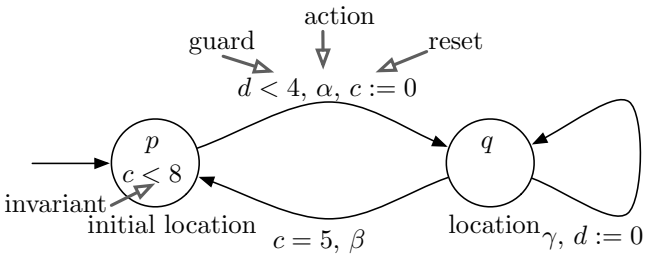

Figure 2: The transition between the location p and q with the label α may take place whenever the clock $d < 4$ and must take place before the clock $c \geq 8$. Once this transition occurs, the clock c resets to 0.

there are two clocks denoted by c and d, and actions designated by α, β, and γ. The transition between location p and q may take place whenever the clock $d < 4$ and must take place for the clock $c \geq 8$. Once this transition occurs, the clock c resets to 0.

We follow Alur and Dill (1994) and define a timed automata as follows. A set of diagonal-free clock constraints $\Psi(C)$ for the set C of clocks contains all invariants and guards of the timed automaton. Consequently, it is described by the following grammar

$$\psi ::= c \bowtie k | \psi_1 \wedge \psi_2, \tag{5a}$$

where

$$c \in C,\ k \in \mathbb{R}_+,\ \text{and}\ \ \bowtie \in \{\leq, <, =, >, \geq\}. \tag{5b}$$

Note that the clock constraint k should usually be a rational number, but in this paper, no effort is made to convert the clock constraints into rational numbers. However, any real number can be approximated by a rational number with an arbitrary small error $\epsilon > 0$.

Definition 2 (Timed Automaton) *A timed automaton $\mathcal{A}$ is a tuple $(E, E_0, C, \Sigma, I, \Delta)$, where*

- *E is a finite set of locations, and $E_0 \subseteq E$ is the set of initial locations.*
- *C is a finite set of clocks.*
- *Σ is the input alphabet.*
- *$I : E \to \Psi(C)$ assigns invariants to locations, where $\Psi(C)$ is the set of all clock constraints in (5).*
- *$\Delta \subseteq E \times \Psi(C) \times \Sigma \times 2^C \times E$ is a finite set of transition relations. A transition relation is a tuple $(e, G_{e\to e'}, \sigma, R_{e\to e'}, e')$ which assigns an edge between two locations, where e is the source location, e' is the destination location, $G_{e\to e'} \in \Psi(C)$ is the guard set, σ is a symbol in the alphabet Σ, and $R_{e\to e'} \subseteq C$ is a subset of clocks.*

To define the semantics of a timed automaton, we adopt the notion of clock valuation Fahrenberg et al. (2010).

Definition 3 (Clock Valuation) *A clock valuation on a set of clocks C is a mapping $v : C \to \mathbb{R}_+$. The initial valuation v_0 is given by $v_0(c) = 0$ for all $c \in C$. For a valuation v, $d \in \mathbb{R}_+$, and $R \subseteq C$, the valuations $v + d$ and $v[R]$ are defined as*

$$(v + d)(c) = v(c) + d, \tag{6a}$$

$$v[R](c) = \begin{cases} 0 & \text{for } c \in R, \\ v(c) & \text{otherwise.} \end{cases} \tag{6b}$$

We shall denote the set of maps $v : C \to \mathbb{R}_+$ by $\mathbb{R}_+^C$. This notation indicates that we identify a valuation v with C-tuples of nonnegative reals in $\mathbb{R}_+^{\#C}$, where $\#C$ is the number of elements in C. Notice also that this notion is consistent with 2^C denoting the set of subsets of C. Indeed, if 2 denotes the set consisting of two elements, say $\{0, 1\}$, then $e \in 2^C$ is identified with $e^{-1}(1) \subseteq C$.

Definition 4 (Semantics of Clock Constraint) *A clock constraint in $\Psi(C)$ is a set of clock valuations $\{v : C \to \mathbb{R}_+\}$ given by*

$$[\![c \bowtie k]\!] = \{v : C \to \mathbb{R}_+ | v(c) \bowtie k\} \tag{7a}$$

$$[\![\psi_1 \wedge \psi_2]\!] = [\![\psi_1]\!] \cap [\![\psi_2]\!]. \tag{7b}$$

For convenience we denote $v \in [\![\psi]\!]$ by $v \models \psi$.

Definition 5 (Semantics of Timed Automaton) *The semantics of a timed automaton $\mathcal{A} = (E, E_0, C, \Sigma, I, \Delta)$ is the transition system $[\![\mathcal{A}]\!] = (S, S_0, \Sigma \cup \mathbb{R}_+, T_\mathrm{s} \cup T_\mathrm{d})$, where S is the set of states*

$$S = \{(e, v) \in E \times \mathbb{R}_+^C |\ v \models I(e)\},$$

$S_0 \subseteq S$ is the set of initial states

$$S_0 = \{(e, v) \in E_0 \times \mathbb{R}_+^C |\ v = v_0\},$$

and $T_\mathrm{s} \cup T_\mathrm{d}$ is the union of the following sets of transitions

$$T_\mathrm{s} = \{(e, v) \xrightarrow{\sigma} (e', v') |\ \exists (e, G_{e\to e'}, \sigma, R_{e\to e'}, e') \in \Delta \text{ such that } v \models G_{e\to e'} \text{ and } v' = v[R_{e\to e'}]\},$$

$$T_\mathrm{d} = \{(e, v) \xrightarrow{d} (e, v + d) |\ \forall d' \in [0, d] : v + d' \models I(e)\}.$$

Hence, the semantics of a timed automaton is a transition system that comprises of an infinite number of states: product of E and $\mathbb{R}_+^C$ and two types of transitions: the transition set T_s between discrete states with possibly a reset of clocks belonging to a subset

$R_{e\to e'}$, and the transition set T_{d} which corresponds to time passing within the invariant $I(e)$.

The analog to the solution (2) of an autonomous differential equation (1) is a run of a timed automaton, which is define below.

Definition 6 (Run of Timed Automaton) *A run of a timed automaton $\mathcal{A}$ (with semantics $[\![\mathcal{A}]\!]$) is a possibly infinite sequence of alternations between time steps and discrete steps of the following form*

$$(e_0, v_0) \xrightarrow{d_1} (e_0, v_1) \xrightarrow{\sigma_1} (e_1, v_2) \xrightarrow{d_2} \dots \tag{8}$$

where $d_i \in \mathbb{R}_+$ and $\sigma_i \in \Sigma$.

In Definition 6, by forcing alternation of time and discrete steps, the time step d_i is the maximal time step between the discrete steps σ_{i-1} and σ_i.

Example 1 *This example clarifies the semantics of an automaton. A timed automaton with two locations and two clocks is illustrated in Figure 2. All runs of the timed automaton start in the location p, and the initial valuation of all clocks is zero. Furthermore, the time between an action α (a transition decorated by the label α) and an action β is 5 time units. There are infinitely many different runs of the timed automaton, and a few examples are*

$$\text{Run 1}: (p,(0,0)) \xrightarrow{1} (p,(1,1)) \xrightarrow{\alpha} (q,(0,1)) \xrightarrow{2}$$
$$(q,(2,3)) \xrightarrow{\gamma} (q,(2,0)) \xrightarrow{3} (q,(5,3)) \xrightarrow{\beta} (p,(5,3))$$
$$\text{Run 2}: (p,(0,0)) \xrightarrow{3} (p,(3,3)) \xrightarrow{\alpha} (q,(0,3)) \xrightarrow{4}$$
$$(q,(4,7)) \xrightarrow{\gamma} (q,(4,0)) \xrightarrow{1} (q,(5,1)) \xrightarrow{\beta} (p,(5,1)).$$

A vital object for studying the behavior of any dynamical system is its trajectory. For this purpose, we have already defined a run of $\mathcal{A}$ in Definition 6; however, more convenient for the study of continuous behavior of a timed automaton is a trajectory, see Definition 7. At the outset, we bring in a concept of a time domain.

In the following, we denote sets of the form $\{a, \dots\}$ with $a \in \mathbb{Z}_+$ as $\{a, \dots, \infty\}$. Let $k \in \mathbb{N} \cup \{\infty\}$; a subset $\mathcal{T}_k \subset \mathbb{Z}_+ \times \mathbb{R}_+$ will be called a time domain if there exists an increasing sequence $\{t_i\}_{i\in\{0,\dots,k\}}$ in $\mathbb{R}_+ \cup \{\infty\}$ such that

$$\mathcal{T}_k = \bigcup_{i\in\{1,\dots,k\}} \{i\} \times T_i$$

where $T_i = [t_{i-1}, t_i]$ if $i \in \{1, \dots, k-1\}$, and

$$T_k = \begin{cases} [t_{k-1}, t_k] & \text{if } t_k < \infty \\ [t_{k-1}, \infty[& \text{if } t_k = \infty. \end{cases}$$

Note that $T_i = [t_{i-1}, t_i]$ for all i if $k = \infty$. We say that the time domain is infinite if $k = \infty$ or $t_k = \infty$. The sequence $\{t_i |\ i \in \{0, \dots, k\}\}$ corresponding to a time domain will be called a switching sequence.

We define two projections $\pi_1 : E \times \mathbb{R}_+^C \to E$ and $\pi_2 : E \times \mathbb{R}_+^C \to \mathbb{R}_+^C$ by $\pi_1(e, v) = e$ and $\pi_2(e, v) = v$.

Definition 7 (Trajectory) *A trajectory of the timed automaton $\mathcal{A}$ (with semantics $[\![\mathcal{A}]\!]$) is a pair $(\mathcal{T}_k, \gamma)$ where $k \in \mathbb{N} \cup \{\infty\}$ is fixed, and*

- *$\mathcal{T}_k \subset \mathbb{Z}_+ \times \mathbb{R}_+$ is a time domain with corresponding switching sequence $\{t_i |\ i \in \{0, \dots, k\}\}$,*
- *$\gamma : \mathcal{T}_k \to S$ is continuous and satisfies:*
  1. *For each $i \in \{1, \dots, k-1\}$, there exists $\sigma \in \Sigma$ such that*
  $$\gamma(i, t_i) \xrightarrow{\sigma} \gamma(i+1, t_i) \in T_{\mathrm{s}}$$
  2. *For each $i \in \{1, \dots, k-1\}$*
  $$\pi_2(\gamma(i, T_i)) \subset [\![I(\pi_1(\gamma(i, t_i)))]\!].$$

A trajectory at (e, v) (with $v \models I(e)$) is a trajectory $(\mathcal{T}_k, \gamma)$ with $\gamma(1, t_0) = (e, v)$.

We define a discrete counterpart of the flow map.

Definition 8 (Flow Map of Timed Automaton) *The flow map of a timed automaton $\mathcal{A}$ (with semantics $[\![\mathcal{A}]\!]$) is a multivalued map*

$$\phi_{\mathcal{A}} : \mathbb{R}_+ \times S_0 \to 2^S,$$

defined by $(e', v') \in \phi_{\mathcal{A}}(t; e, v)$ if and only if there exists a trajectory $(\mathcal{T}_k, \gamma)$ at (e, v) such that $t = t_k - t_0$ and $(e', v') = \gamma(k, t_k)$.

It will be instrumental to define a discrete flow map which forgets the valuation of the clocks

$$\Phi_{\mathcal{A}} : \mathbb{R}_+ \times E_0 \to 2^E,\ \Phi_{\mathcal{A}}(t, e) = \pi_1 \circ \phi_{\mathcal{A}}(t; e, v_0)$$

In other words, $\Phi_{\mathcal{A}}$ is defined by: $e' \in \Phi_{\mathcal{A}}(t, e)$ if and only if there exists a run (8) of $[\![\mathcal{A}]\!]$ initialized in (e, v_0) that reaches the location e' at time $t = \sum_i d_i$.

Example 2 (Continuation of Example 1) *In this example, the time domain $\mathcal{T}_k$ and trajectory of Run 1 in Example 1 are elucidated. The time domain is*

$$\mathcal{T}_k = \{1\} \times [0, 1] \cup \{2\} \times [1, 3] \cup \{3\} \times [3, 6] \cup \{4\} \times [6, 6]. \tag{9}$$

From the time domain it is seen that there are three discrete switches and the total time of the run is 6 time

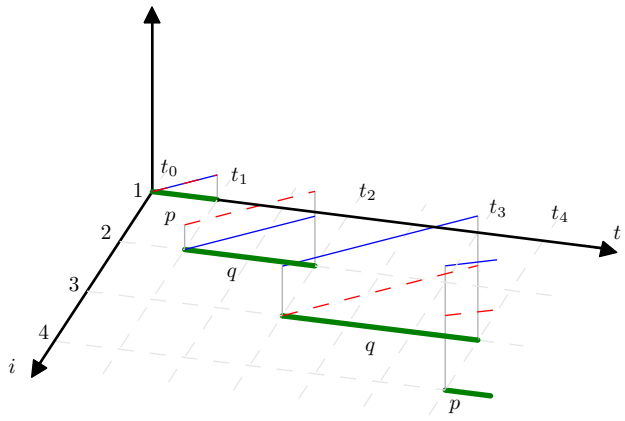

Figure 3: Trajectory of Run 1 in Example 1.

units. The trajectory of the run is shown in Figure 3. To visualize the trajectory, the valuation of the clock c is illustrated by a blue line and the valuation of clock d is illustrated by a red dashed line. Furthermore, the location, which the system is in at a given time, is indicated by its name.

As for a dynamical system, equipped with the notion of the discrete flow map, we define a reachable set.

Definition 9 (Reachable set of Timed Autom.) *The reachable locations of a system $\mathcal{A}$ from a set of initial locations $E_0 \subseteq E$ on the time interval $[t_1, t_2]$ is defined as*

$$\Phi_{\mathcal{A}}([t_1, t_2], E_0) \equiv \bigcup_{(t,e)\in[t_1,t_2]\times E_0} \Phi_{\mathcal{A}}(t, e). \qquad (10)$$

Thus far, we have avoided any explanation of the role of labels in a timed automaton. In fact, the meaning of labeling first becomes apparent in a network of agents, where each agent is modeled as a timed automaton. Subsequently, the transitions of two automata with the same labels can be synchronized. We adopt the definition of the product of timed automata from Alur (1999).

Definition 10 (Product of Timed Automaton) *Let $\mathcal{A}^1 = (E^1, E_0^1, C^1, \Sigma^1, I^1, \Delta^1)$ and $\mathcal{A}^2 = (E^2, E_0^2, C^2, \Sigma^2, I^2, \Delta^2)$. Suppose that $C^1 \cap C^2 = \emptyset$. Then the product $\mathcal{A}^1||\mathcal{A}^2$ is the time automaton $(E, E_0, C, \Sigma, I, \Delta)$, where $E \equiv E^1 \times E^2$, $E_0 \equiv E_0^1 \times E_0^2$, $C \equiv C^1 \cup C^2$, $\Sigma \equiv \Sigma^1 \cup \Sigma^2$, along with I and Δ defined as follows*

- $I : E \to \Psi(C)$, $I(e^1, e^2) \equiv I(e^1) \wedge I(e^2)$;
- $\Delta \subseteq E \times \Psi(C) \times \Sigma \times 2^C \times E$ *is defined by*

1. *For every* $\sigma \in \Sigma_1 \cup \Sigma_2$, *for every* $(e^1, G^1, \sigma, R^1, f^1) \in \Delta^1$ *and for every* $(e^2, G^2, \sigma, R^2, f^2) \in \Delta^2$

$$((e^1, e^2), G^1 \wedge G^2, \sigma, R^1 \cup R^2, (f^1, f^2)) \in \Delta.$$

2. *For every* $\sigma \in \Sigma^1 \setminus \Sigma^2$, *for every* $(e^1, G, \sigma, R, f^1) \in \Delta^1$ *and for every* $e^2 \in E^2$

$$((e^1, e^2), G, \sigma, R, (f^1, e^2)) \in \Delta.$$

3. *For every* $\sigma \in \Sigma^2 \setminus \Sigma^1$, *for every* $(e^2, G, \sigma, R, f^2) \in \Delta^2$ *and for every* $e^1 \in E^1$

$$((e^1, e^2), G, \sigma, R, (e^1, f^2)) \in \Delta.$$

In Section 4, we will use the notion of an isomorphism of timed automata. We say that two automata are isomorphic if they differ merely by the names of their states.

Definition 11 (Isomorphism of Timed Automata) *Two timed automata $(E, E_0, C, \Sigma, I, \Delta)$ and $(E', E_0', C, \Sigma, I', \Delta')$ (with the same sets of labels and clocks) are isomorphism if there is a bijective function $F : S \to S'$ such that*

1. $F(S_0) = S_0'$,
2. *for all* $s \in S$, $I(s) = I'(F(s))$,
3. $(e, G_{e\to e'}, \sigma, R_{e\to e'}, e') \in \Delta$ *if and only if* $(F(e), G_{e\to e'}, \sigma, R_{e\to e'}, F(e')) \in \Delta'$.

Our final remark is about the reachability problem of timed automata. For a given timed automaton $\mathcal{A}$, a set of terminal locations F, and a time interval $[t_1, t_2]$, we ask the question if $\Phi_{\mathcal{A}}([t_1, t_2], E_0) \cap F$ is nonempty. Nonetheless, to study reachability by combinatorial methods such as formal verification methods, the set of states of the semantics $[\![\mathcal{A}]\!]$ of $\mathcal{A}$ is to be finite. At the same time, the choice of clock constraints indicates that it is only possible to determine if the clocks are equal, less or greater to each other. Consequently, Alur (1999) introduces the concept of region automaton. In this work, we explain this abstraction geometrically. In short, the set $\mathbb{R}_+^C$ is partitioned by a complex K consisting of all the faces of the cubes (of dimension $n \equiv \#C$) in $\{c_\alpha |\ \alpha \in \mathbb{N}^n\}$, where $c_\alpha \equiv \{x = (x_1, \ldots, x_n) \in \mathbb{R}_+^n |\ \alpha_i \leq x_i \leq \alpha_i + 1\}$. Figure 4 illustrates the partitioning for two clocks.

4 Abstractions of Dynamical Systems

In this section, we develop a concept of an abstraction of the dynamical system Γ. It consists of a fine number

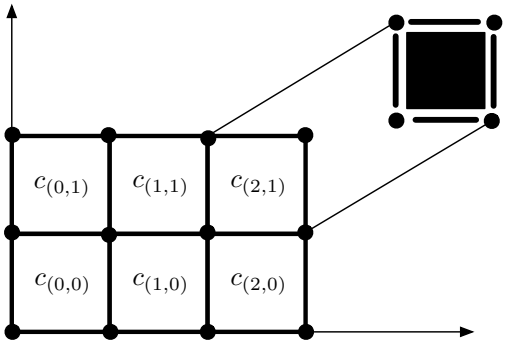

Figure 4: The valuations of two clocks are partitioned into 6 squares, 17 edges, and 12 vertices.

of sets $E \equiv \{e_\lambda |\ \lambda \in \Lambda\}$, which we call cells. The cells cover the state space X

$$X = \bigcup_{\lambda \in \Lambda} e_\lambda.$$

To the partitioning E, we associate an abstraction function, which to each point in the state space associates the cells that this point belongs to.

Definition 12 (Abstraction Function) *Let $E \equiv \{e_\lambda |\ \lambda \in \Lambda\}$ be a finite partition of the state space $X \subseteq \mathbb{E}$. An abstraction function for E is the multi-valued function defined by*

$$\alpha_E : X \to 2^E,\ \alpha_E(x) = \{e \in E |\ x \in e\}.$$

At last, we are able to formulate the objectives of this work rigorously. For a given dynamical system Γ, we want to simultaneously devise a partitioning E of the state space X and create a time automaton $\mathcal{A}$ with locations E such that

1. The abstraction is **sound** on an interval $[t_1, t_2]$:

   $$\alpha_E \circ \Phi_\Gamma(X_0, t) \subseteq \Phi_\mathcal{A}(\alpha_E(X_0), t),\ \text{for all } t \in [t_1, t_2]$$

2. The abstraction is **complete** on an interval $[t_1, t_2]$:

   $$\alpha_E \circ \Phi_\Gamma(X_0, t) = \Phi_\mathcal{A}(\alpha_E(X_0), t) \text{ for all } t \in [t_1, t_2].$$

Figure 5 illustrates the reachable set of a dynamical system, along with reachable sets of a sound abstraction (left) and a complete abstraction (right).

4.1 Partitioning the State Space

This subsection presents the proposed partitioning. The cells of the partition are generated by intersections of sublevel sets of functions. To generate sound and complete abstractions, we use functions, whose sublevel sets are positively invariant. We call such functions partitioning functions.

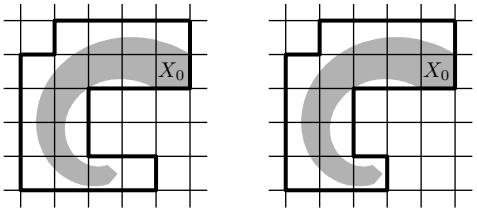

Figure 5: Reachable set of a dynamical system (shaded area), and reachable sets of automata (cells within bold lines). In the left figure, the reachable set of the automaton includes more cells than the ones reached by the dynamical system, i.e., the abstraction is sound. In the right figure, the reachable set of the automaton includes only the cells that are reached by the dynamical system, i.e., the abstraction is complete.

To this end, we define a slice as the set-difference of invariant sets.

Definition 13 (Slice) *A nonempty set S is a slice if there exist two open sets A_1 and A_2 such that*

1. *A_1 and A_2 are positively invariant,*
2. *A_1 is a proper subset of A_2, and*
3. *$S = \text{cl}(A_2 \backslash A_1)$.*

It is seen that since A_1 and A_2 are positively invariant sets, a trajectory initialized in S can propagate to A_1, but a solution initialized in A_1 cannot propagate to S. This implies that, via these invariants, we can to some extend study the possible trajectories of a dynamical system. We will adopt the convention that $\emptyset$ is a positively invariant set of any dynamical systems.

Example 3 *Consider two second order dynamical systems $\Gamma_i = (\mathbb{E}^2, x \mapsto L_i x)$ with*

$$L_1 = \begin{bmatrix} -\lambda_1 & 0 \\ 0 & -\lambda_2 \end{bmatrix},\ L_2 = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix},\ \lambda_1, \lambda_2 > 0.$$

Let $D = \{x |\ x < 1\}$, and suppose $A_1 = \emptyset$, $A_2 = D$. Hence, A_1 and A_2 are positively invariant sets for Γ_1. Therefore, $S = \text{cl}(A_2 \backslash A_1) = \text{cl}(D)$ is a slice.

For Γ_2, let $A_1 = \mathbb{E} \backslash \text{cl}(D)$ and $A_2 = \mathbb{E}$. In like fashion, A_1 and A_2 are positively invariant sets for Γ_2, and $S = \text{cl}(A_2 \backslash A_1) = \text{cl}(D)$ is a slice.

To devise a partition of a state space, we need to define finite collections of slices. These collections are called slice-families.

Definition 14 (Slice-Family) *Let*

$$A_0 \subset A_1 \subset \cdots \subset A_k$$

be a collection of positive invariant sets of a dynamical system $\Gamma = (X, \xi)$ *with* $X \subseteq A_k$.
We say that the collection

$$\mathcal{S} \equiv \{S_i = \mathrm{cl}(A_i \backslash A_{i-1}) |\ i = 1, \dots k\}$$

is a slice-family generated by the sets $\{A_i |\ i = 1, \dots, k\}$ *or just a slice-family.*

We associate a function to each slice-family $\mathcal{S}$ to provide a simple way of describing the boundary of a slice. Such a function is called a partitioning function.

Definition 15 (Partitioning Function) *Let* $\mathcal{S}$ *be a slice-family generated by the sets* $\{A_i |\ i = 1, \dots, k\}$, *then a continuous function* $\varphi : \mathbb{E} \to \mathbb{R}$ *smooth on* $\mathbb{E} \backslash \mathrm{Cr}(\xi)$ *is a partitioning function for* $\mathcal{S}$ *if there is a sequence*

$$a_0 < \ldots < a_k, \quad a_i \in \mathbb{R} \cup \{-\infty, \infty\}$$

of regular values of φ *such that*

$$\mathrm{cl}(A_i) = \varphi^{-1}([a_{i-1}, a_i]). \tag{11}$$

We remark that by regular level set theorem, for $a_i \in \mathbb{R}$, the boundary $\varphi^{-1}(a_i)$ of A_i is an embedded smooth submanifold of $\mathbb{E}$ of co-dimension 1 Tu (2008).

As stated in the beginning of the section, we will create cells that cover the entire state space. They are obtained by intersecting slices. To ensure robustness of the partition, it is important that the slices intersect transversally. The robustness of a transversal intersection is readily seen from the definition of transversal intersection Hirsch (1976).

Definition 16 (Transversal Intersection) *Suppose that* N_1 *and* N_2 *are embedded submanifolds of* M. *We say that* N_1 *intersects* N_2 *transversally if, whenever* $p \in N_1 \cap N_2$, *we have* $T_p(N_1) + T_p(N_2) = T_p(M)$. *(The sum is not direct, just the set of sums of vectors, one from each of the two subspaces of the tangent space* $T_p(M)$.*)*

The left subplot of Figure 6 illustrates level sets of two partitioning functions (hence two embedded submanifolds of $\mathbb{E}^2$). They intersect at the point p, and their tangents (black lines) are identical. This implies that their tangent vectors only span one dimension at p, i.e., $T_p(N_1) + T_p(N_2) \neq T_p(M)$. Therefore, this intersection is not transversal. Note that with an arbitrary small perturbation, the intersection of the two level sets will be empty (This perturbation is given by a smooth map, see Theorem 2.1 in Hirsch (1976)). Therefore, this partition is not robust.

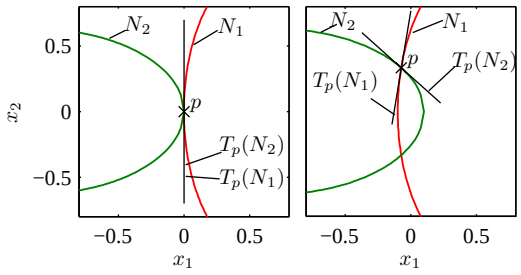

Figure 6: The left subplot shows an intersection that is not transversal; whereas, the right subplot shows aa transversal intersection of two level sets.

In the right subplot Figure 6, two level sets intersecting at point p are illustrated. Their tangent vectors (black lines) span $\mathbb{E}^2$, i.e., the level sets intersect transversally. Note that two manifolds that do not intersect are also transversal.

We define a transversal intersection of slices as follows.

Definition 17 (Transversal Intersection of Slices) *We say that the slices* S_1 *and* S_2 *intersect each other transversally and write*

$$S_1 \pitchfork S_2 = S_1 \cap S_2 \tag{12}$$

if their boundaries, $\mathrm{bd}(S_1)$ *and* $\mathrm{bd}(S_2)$, *intersect each other transversally.*

Cells are generated via intersecting slices. We denote cardinality (number of elements) of a finite set $\mathcal{S}$ by $|\mathcal{S}|$.

Definition 18 (Extended Cell) *Let* $\mathcal{S} = \{\mathcal{S}^i | i \in \{1, \dots, k\}\}$ *be a collection of* k *slice-families and let* $\mathcal{G}(\mathcal{S}) \equiv \{1, \dots, |\mathcal{S}^1|\} \times \cdots \times \{1, \dots, |\mathcal{S}^k|\} \subset \mathbb{N}^k$. *Denote the* j^{th} *slice in* $\mathcal{S}^i$ *by* S^i_j *and let* $g \in \mathcal{G}(\mathcal{S})$. *Then*

$$e_{\mathrm{ex},g} \equiv \pitchfork_{i=1}^{k} S^i_{g_i} \tag{13}$$

where g_i *is the* i^{th} *component of the vector* g. *Any nonempty set* $e_{\mathrm{ex},g}$ *is called an extended cell of* $\mathcal{S}$.

The cells in (13) are denoted by extended cells, since the transversal intersection of slices may form multiple disjoint sets in the state space. It is desired to have cells, which are connected. Therefore, the following is defined.

Definition 19 (Cell) *A cell of* $\mathcal{S}$ *is a connected component of an extended cell of* $\mathcal{S}$

$$\bigcup_h e_{(g,h)} = e_{\mathrm{ex},g}, \tag{14a}$$

where

$$e_{(g,h)} \cap e_{(g,h')} = \emptyset \quad \forall h \neq h'. \tag{14b}$$

Example 4 *This example illustrates the concepts of extended cells and cells. Figure 7 shows a partition of a two-dimensional state space generated by two slice-families. The intersection of a slice from each slice-families is an extended cell. The shaded area indicates an extended cell that consists of four connected components. Each connected component is a cell.*

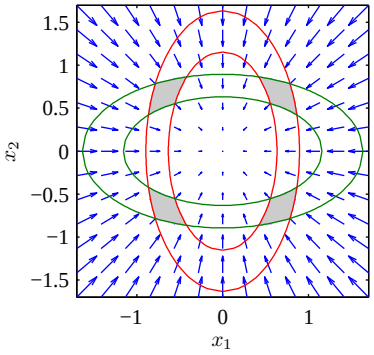

Figure 7: Illustration of an extended cell (shaded area). The extended cell is generated by intersecting slices from two slice-families. The red lines are the boundaries of the slice from the first slice-family; accordingly, the green lines are the boundaries of the slice from the second slice-family.

A finite partition based on the transversal intersection of slices is defined in the following.

Definition 20 (Finite Partition) *Let $\mathcal{S}$ be a collection of slice-families, $\mathcal{S} = \{\mathcal{S}^i | i \in \{1,\dots,k\}\}$. We define a finite partition $E(\mathcal{S})$ by*

$$e \in E(\mathcal{S}) \tag{15}$$

if and only if e is a cell of $\mathcal{S}$.

We propose to use Lyapunov functions as partitioning functions, to obtain robustness of the partition. The robustness is secured as the vector field is transversal to the boundaries of the cells. This implies that there exists an arbitrary small perturbation of the vector field, such that it is still transversal to the boundary of the cells. The following definition of Lyapunov function origin from Meyer (1968).

Definition 21 (Lyapunov Function) *Let X be an open connected subset of $\mathbb{E}^n$. Suppose $\xi : X \to \mathbb{E}^n$ is continuous, and recall that $\mathrm{Cr}(\xi)$ denotes the set of critical points of ξ. Then a real non-degenerate (see (Matsumoto, 2002, p. 1)) differentiable function $\varphi : X \to \mathbb{R}$ is said to be a Lyapunov function for ξ if*

$$p \text{ is a critical point of } \xi \Leftrightarrow p \text{ is a critical point of } \varphi$$

$$\dot{\varphi}(x) \equiv \sum_{j=1}^{n} \frac{\partial \varphi}{\partial x_j}(x)\xi^j(x) \tag{16a}$$

$$\dot{\varphi}(x) < 0 \ \ \forall x \in X \backslash \mathrm{Cr}(\xi) \tag{16b}$$

and there exists $\alpha > 0$ and an open neighborhood of each critical point $p \in \mathrm{Cr}(\xi)$, where

$$||\dot{\varphi}(x)|| \geq \alpha ||x - p||^2. \tag{17}$$

Notice that we only require the vector field to be transversal to the level curves of a Lyapunov function φ, i.e., $\dot{\varphi}(x) = \langle \nabla\varphi(x), \xi(x)\rangle < 0$ for all $x \in X \backslash \mathrm{Cr}(\xi)$, and does not use Lyapunov functions in the usual sense, where the existence of a Lyapunov function implies stability, but uses a more general notion from Meyer (1968). Assume that a Lyapunov function $\varphi(x)$ is positive definite, then its sublevel sets are positively invariant.

A partitioning function φ^i for a slice-family $\mathcal{S}^i$ that is Lyapunov will be called a Lyapunov function for $\mathcal{S}^i$.

4.2 Generation of Abstraction

This subsection explains how a timed automaton $\mathcal{A}$ is generated from a finite partition $E(\mathcal{S})$ of the state space of a system $\Gamma = (X, \xi)$.

Definition 22 (Generation of Timed Automaton) *Let $\mathcal{S} = \{\mathcal{S}^i | \ i \in \{1,\dots,k\}\}$ be a finite collection of slice-families, and $\mathcal{T} = \{(\underline{t}^i_{g_i}, \overline{t}^i_{g_i}) \in \mathbb{R}^2_+ | \ i \in \{1,\dots,k\}, g_i \in \{1,\dots,|\mathcal{S}^i|\}\}$. Then the timed automaton $\mathcal{A}(\mathcal{S},\mathcal{T}) = (E, E_0, C, \Sigma, I, \Delta)$ is defined by*

- ***Locations:*** *The locations of $\mathcal{A}$ are given by*

$$E = E(\mathcal{S}). \tag{18}$$

  This means that a location $e_{(g,h)}$ is identified with the cell $e_{(g,h)} = \alpha^{-1}_{E(\mathcal{S})}(\{e_{(g,h)}\})$ of the partition $E(\mathcal{S})$, see Definition 12.

- ***Clocks:*** *The set of clocks is*

$$C = \{c^i | i \in \{1,\dots,k\}\}$$

  .

- ***Alphabet:*** *The alphabet is*

$$\Sigma = \{\sigma^i | i \in \{1,\dots,k\}\}.$$

- ***Invariants:*** *In each location $e_{(g,h)}$, we impose an invariant*

$$I(e_{(g,h)}) = \bigwedge_{i=1}^{k} c^i \leq \bar{t}^i_{g_i}. \tag{19}$$

- ***Transition relations:*** *If a pair of locations $e_{(g,h)}$ and $e_{(g',h')}$ satisfy the following two conditions*

  1. *$e_{(g,h)}$ and $e_{(g',h')}$ are adjacent, that is $e_{(g,h)} \cap e_{(g',h')} \neq \emptyset$, and*
  2. *$g'_i \leq g_i$ for all $i \in \{1,\ldots,k\}$.*

  Then there is a transition relation

$$\delta_{(g,h)\to(g',h')} = (e_{(g,h)}, G_{(g,h)\to(g',h')}, \sigma^i, R_{(g,h)\to(g',h')}, e_{(g',h')}), \tag{20a}$$

where

$$G_{(g,h)\to(g',h')} = \bigwedge_{i=1}^{k} \begin{cases} c^i \geq \underline{t}^i_{g_i} & \text{if } g_i - g'_i = 1 \\ c^i \geq 0 & \text{otherwise.} \end{cases} \tag{20b}$$

Note that $g_i - g'_i = 1$ whenever a transition labeled σ^i is taken.

Let $i \in \{1,\ldots,k\}$. We define $R_{(g,h)\to(g',h')}$ by

$$c^i \in R_{(g,h)\to(g',h')} \tag{20c}$$

iff $g_i - g'_i = 1$.

The semantics of $(\underline{t}^i_{g_i}, \bar{t}^i_{g_i}) \in \mathcal{T}$ is the pair of a lower- and an upper-bound on the time for any trajectory to traverse the slice $S^i_{g_i}$.

If the set $\mathcal{S}$ is a singleton, i.e., $\mathcal{S} = \{\mathcal{S}_1\}$ then by slightly abusing the notation, we write $\mathcal{A}\left(\mathcal{S}_1, (\underline{t}, \bar{t})\right)$ instead of $\mathcal{A}\left(\{\mathcal{S}_1\}, \{(\underline{t}, \bar{t})\}\right)$.

Definition 23 *A timed automaton $\mathcal{A}_{\text{ex}}(\mathcal{S},\mathcal{T})$ has locations given by*

$$E = E_{\text{ex}}(\mathcal{S}) \tag{21}$$

where a location $e_{\text{ex},g} \in E_{\text{ex}}(\mathcal{S})$ is associated with the extended cell $e_{\text{ex},g}$ generated by the slice-family $\mathcal{S}$; hence, $e_{\text{ex},g} = \alpha^{-1}_{E_{\text{ex}}(\mathcal{S})}(\{e_{\text{ex},g}\})$.

4.3 Properties of the Abstraction

In this subsection, we present some compositionality results, which enables verification of high dimensional systems. Furthermore, sufficient conditions for soundness and completeness are presented. Proofs of the propositions presented in this subsection can be found in Sloth and Wisniewski (2011).

For a collection $\mathcal{S} = \{\mathcal{S}^i |\ i \in \{1,\ldots,k\}\}$ of slice-families, a product of timed automata $\mathcal{A}(\mathcal{S}^i)$ for $i \in \{1,\ldots,k\}$ is similar to the intersection of slices in the slice-families $\mathcal{S}^i$. Therefore, the intersection of slices should be nonempty to let the locations of the timed automaton $\mathcal{A}_{\text{ex}}(\mathcal{S})$ be such a product, as stated in Proposition 1.

Proposition 1 *Let $\mathcal{S} = \{\mathcal{S}^i | i \in \{1,\ldots,k\}\}$ be a collection of slice-families , and $\mathcal{T} = \{(\underline{t}^i_{g_i}, \bar{t}^i_{g_i}) \in \mathbb{R}^2_+ |\ i \in \{1,\ldots,k\},\ g_i \in \{1,\ldots,|\mathcal{S}^i|\}\}$. Suppose that for each $i,j \in \{1,\ldots,k\}$, and for each $g_i \in \{1,\ldots,|\mathcal{S}^i|\}$ and $g_j \in \{1,\ldots,|\mathcal{S}^j|\}$, we have*

$$S^i_{g_i} \pitchfork S^j_{g_j} \neq \emptyset \tag{22}$$

(the intersection is transversal and nonempty).

Then, $\mathcal{A}_{\text{ex}}(\mathcal{S},\mathcal{T})$ is isomorphic to the product of k timed automata,

$$\mathcal{A}\left(\mathcal{S}^1, (\underline{t}^1_{g_1}, \bar{t}^1_{g_1})\right) || \ldots || \mathcal{A}\left(\mathcal{S}^k, (\underline{t}^k_{g_k}, \bar{t}^k_{g_k})\right).$$

The property that $\mathcal{A}_{\text{ex}}(\mathcal{S},\mathcal{T})$ is isomorphic to the product of k timed automata is of paramount importance for computations, since it allows parallel verification of the k timed automata each with only one clock. Furthermore, it makes it possible to sequentially add slice-families to the abstraction, to replace, and to refine slice-families to improve the accuracy of the abstraction.

The product of timed automata also allows the sequential verification of the abstraction. We show this in terms of safety in the following.

Corollary 1 *Suppose the premises of Proposition 1 hold. If the timed automaton*

$$\mathcal{A}\left(\mathcal{S}^1, (\underline{t}^1_{g_1}, \bar{t}^1_{g_1})\right) || \ldots || \mathcal{A}\left(\mathcal{S}^j, (\underline{t}^j_{g_j}, \bar{t}^j_{g_j})\right)$$

is safe for some $j \in \{1,\ldots,k\}$. Then, $\mathcal{A}_{\text{ex}}(\mathcal{S},\mathcal{T})$ is also safe.

Sufficient conditions for soundness and completeness of an abstraction are formulated in the following.

Proposition 2 (Sufficient Cond. for Soundness) *Let $\mathcal{S} = \{\mathcal{S}^i | i \in \{1,\ldots,k\}\}$ be a collection of slice-families. For $i \in \{1,\ldots,k\}$, let φ^i be a Lyapunov function for the slice-family $\mathcal{S}^i$. Suppose $\mathcal{T} = \{(\underline{t}^i_{g_i}, \bar{t}^i_{g_i}) \in \mathbb{R}^2_+ |\ i \in \{1,\ldots,k\},\ g_i \in \{1,\ldots,|\mathcal{S}^i|\}\}$ with*

$$\underline{t}^i_{g_i} \leq \frac{a^i_{g_i} - a^i_{g_i-1}}{\sup\{|\dot{\varphi}^i(x)| \in \mathbb{R}_+ | x \in S^i_{g_i}\}} \tag{23a}$$

$$\bar{t}^i_{g_i} \geq \frac{a^i_{g_i} - a^i_{g_i-1}}{\inf\{|\dot{\varphi}^i(x)| \in \mathbb{R}_+ | x \in S^i_{g_i}\}}, \tag{23b}$$

where $\dot{\varphi}^i(x)$ is defined as shown in (16a). Then a timed automaton $\mathcal{A}(\mathcal{S},\mathcal{T})$ is a sound abstraction of the system $\Gamma = (X,\xi)$.

The sufficient condition states that the abstraction is sound if $\underline{t}^i_{g_i}$ is less than or equal to the time it takes to traverse $S^i_{g_i}$ maintaining a constant speed equal to the largest possible speed within $S^i_{g_i}$. Similarly, $\overline{t}^i_{g_i}$ is to be greater than or equal to the time it takes to traverse $S^i_{g_i}$ maintaining a constant speed equal to the smallest possible speed within $S^i_{g_i}$.

Proposition 3 (Suff. Cond. for Completeness) *Let $\mathcal{S} = \{\mathcal{S}^i | i \in \{1,\dots,k\}\}$ be a collection of slice-families, and let*

$$S^i_{g_i} = (\varphi^i)^{-1}([a^i_{g_i-1}, a^i_{g_i}]). \tag{24}$$

If the following two conditions are satisfied

1. *for any $g \in \mathcal{G}(\mathcal{S})$, recall the definition of $\mathcal{G}(\mathcal{S})$ from Definition 18, with $g_i \geq 2$ there exists a time $t^i_{g_i}$ such that for all $x_0 \in (\varphi^i)^{-1}(a^i_{g_i})$*

$$\phi_\Gamma(t^i_{g_i}, x_0) \in (\varphi^i)^{-1}(a^i_{g_i-1}) \tag{25}$$

and

2. $\overline{t}_{S^i_{g_i}} = \underline{t}_{S^i_{g_i}} = t^i_{g_i}$

then a timed automaton $\mathcal{A}(\mathcal{S},\mathcal{T})$ with $\mathcal{T} = \{(\underline{t}^i_{g_i}, \overline{t}^i_{g_i}) |\ i \in \{1,\dots,k\},\ g_i \in \{1,\dots,|\mathcal{S}^i|\}\}$ is a complete abstraction of Γ.

Equation (25) states that it takes the time $t^i_{g_i}$ for all trajectories of Γ to propagate from $(\varphi^i)^{-1}(a^i_{g_i})$ to $(\varphi^i)^{-1}(a^i_{g_i-1})$ (i.e., $t^i_{g_i}$ is the time to traverse the slice $S^i_{g_i}$). If in addition, the time bounds for both the invariant and guard conditions are the same (i.e., $\overline{t}_{S^i_{g_i}} = \underline{t}_{S^i_{g_i}} = t^i_{g_i}$) then the abstraction is complete. Recall that $\overline{t}$ is used for invariants, while $\underline{t}$ is used for guard conditions.

Proposition 3 is difficult to use for generating partitioning functions. Therefore, the following proposition gives a sufficient condition for satisfying (25), based on the partitioning functions themselves.

Proposition 4 *Let $\mathcal{S} = \{\mathcal{S}^i | i \in \{1,\dots,k\}\}$ be a collection of slice-families. For the system Γ, let $\varphi^i(x)$ be a Lyapunov function for the slice-family $\mathcal{S}^i$ ($i \in \{1,\dots,k\}$), i.e., $S^i_{g_i} = (\varphi^i)^{-1}([a^i_{g_i-1}, a^i_{g_i}])$. If*

$$\varphi^i(x) = \alpha\dot{\varphi}^i(x) \quad \forall x \in \mathbb{R}^n \tag{26}$$

then there exists a time $t^i_{g_i}$ such that for all $x_0 \in (\varphi^i)^{-1}(a^i_{g_i})$ with $g_i \geq 2$

$$\phi_\Gamma(t^i_{g_i}, x_0) \in (\varphi^i)^{-1}(a^i_{g_i-1}). \tag{27}$$

5 Illustrative Example

To illustrate the use of the developed abstraction method, an example is provided. It demonstrates what type of questions can be answered using the proposed abstraction.

In the example, we consider a simple dynamical system, but a quite complicated specification. The system is given by the following third order differential equation

$$\dot{x} = \begin{bmatrix} -0.5387 & -0.2851 & -0.3479 \\ -0.3815 & -0.3010 & -0.2343 \\ -0.0512 & -0.1277 & -0.1078 \end{bmatrix} x. \tag{28}$$

Subsequently, we check if the system satisfies the following specification illustrated in Figure 8: Do all trajectories of the system (28) initialized in X_0 (blue box)

- avoid the unsafe region (red box),
- and reach the goal set (green box) within 10 s and stay there.

To verify this specification, we partition the state space using three quadratic Lyapunov functions $\varphi^i(x) = x^{\mathrm{T}} P^i x$, for $i \in \{1,2,3\}$ and

$$P^1 = \begin{bmatrix} 0.5826 & 0.4020 & 0.4058 \\ 0.4020 & 0.2832 & 0.2811 \\ 0.4058 & 0.2811 & 0.2847 \end{bmatrix} \tag{29a}$$

$$P^2 = \begin{bmatrix} 1.4606 & -1.8855 & 0.2278 \\ -1.8855 & 2.5466 & -0.7147 \\ 0.2278 & -0.7147 & 1.6435 \end{bmatrix} \tag{29b}$$

$$P^3 = \begin{bmatrix} 1.3272 & -2.0953 & 1.6566 \\ -2.0953 & 3.5655 & -3.7219 \\ 1.6566 & -3.7219 & 7.3826 \end{bmatrix}. \tag{29c}$$

The figure does not show the partition, but both the requirements and some trajectories of (28) are depicted.

The analysis of the resulting timed automaton has shown that the specifications are satisfied, as no trajectories reach the red box, and all trajectories reach the green box after 7.7 s. This also complies with the simulated trajectories shown in Figure 8.

6 Conclusions

We have presented a method for abstracting autonomous dynamical systems by timed automata. The method is based on partitioning the state space using positive invariant sets, which are generated by sublevel sets of a family of Lyapunov functions. The proposed method enables formal verification of reachability and timing requirements of a dynamical system. This is

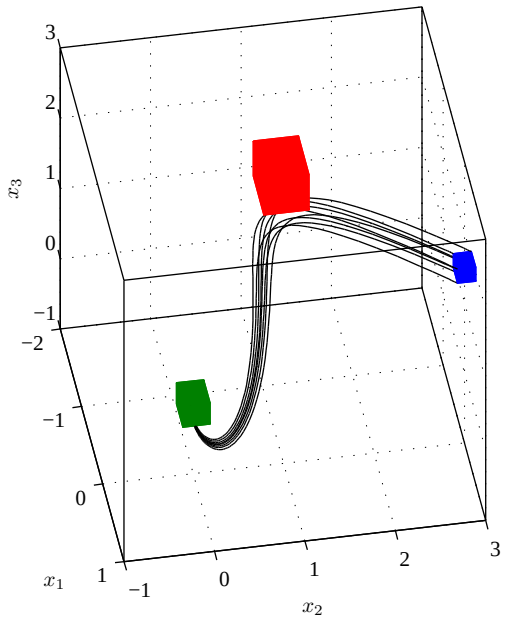

Figure 8: The blue box illustrates the initial states of the system, the red box illustrates the unsafe states, and the green box illustrates the goal states. A set of system trajectories are drawn with black lines.

done by model checking of the generated timed automaton.

The abstraction method is compositional, in the sense that an abstraction of a high-dimensional system can be generated as a product of timed automata each having one clock. This improves the scalability of the method. Furthermore, sufficient conditions for sound and complete abstractions have been presented. These conditions indicate how well the behavior of the abstraction resembles the dynamical system. Finally, an example has been provided to illustrate a specification that is possible to verify using the proposed abstraction method.

Acknowledgments

This work has been supported by MT-LAB, a VKR Centre of Excellence. We would hereby like to express our gratitude for this support.

References

Alur, R. Timed automata. In *Computer aided verification (Trento, 1999)*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer, Berlin, 1999. doi:10.1007/3-540-48683-63.

Alur, R., Courcoubetis, C., and Dill, D. Model-checking for real-time systems. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*. pages 414–425, 1990. doi:10.1109/LICS.1990.113766.

Alur, R. and Dill, D. L. A theory of timed automata. *Theoretical Computer Science*, 1994. 126(2):183–235. doi:10.1016/0304-3975(94)90010-8.

Asarin, E., Dang, T., Frehse, G., Girard, A., Guernic, C. L., and Maler, O. Recent progress in continuous and hybrid reachability analysis. In *Proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design*. Munich, Germany, pages 1582–1587, 2006. doi:10.1109/CACSD-CCA-ISIC.2006.4776877.

Behrmann, G., David, A., and Larsen, K. A tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer Berlin / Heidelberg, 2004. doi:10.1007/978-3-540-30080-9_7.

Fahrenberg, U., Larsen, K., and Thrane, C. Verification, performance analysis and controller synthesis for real-time systems. In *Fundamentals of Software Engineering*, volume 5961 of *Lecture Notes in Computer Science*, pages 34–61. Springer Berlin / Heidelberg, 2010. doi:10.1007/978-3-642-11623-0_2.

Frehse, G. Phaver: Algorithmic verification of hybrid systems past HyTech. In *Hybrid Systems: Computation and Control*, volume 3414 of *Lecture Notes in Computer Science*, pages 258–273. Springer Berlin / Heidelberg, 2005. doi:10.1007/978-3-540-31954-2_17.

Girard, A. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control*, volume 3414 of *Lecture Notes in Computer Science*, pages 291–305. Springer Berlin / Heidelberg, 2005. doi:10.1007/978-3-540-31954-2_19.

Guéguen, H., Lefebvre, M.-A., Zaytoon, J., and Nasri, O. Safety verification and reachability analysis for hybrid systems. *Annual Reviews in Control*, 2009. 33(1):1367–5788. doi:10.1016/j.arcontrol.2009.03.002.

Hirsch, M. W. *Differential Topology*. Springer, Heidelberg, 1976.

Kurzhanski, A. B. and Vályi, I. *Ellipsoidal Calculus for Estimation and Control*. Birkhäuser Boston, 1997.

Maler, O. and Batt, G. Approximating continuous systems by timed automata. In *Formal Methods in Systems Biology*, volume 5054 of *Lecture Notes in*

Computer Science, pages 77–89. Springer Berlin / Heidelberg, 2008. doi:10.1007/978-3-540-68413-8_6.

Matsumoto, Y. *An Introduction to Morse Theory*. American Mathematical Society, 2002.

Meyer, K. R. Energy functions for Morse Smale systems. *American Journal of Mathematics*, 1968. 90(4):1031–1040. doi:10.2307/2373287.

Mitchell, I., Bayen, A., and Tomlin, C. A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on Automatic Control*, 2005. 50(7):947–957. doi:10.1109/TAC.2005.851439.

Prajna, S. Barrier certificates for nonlinear model validation. *Automatica*, 2006. 42(1):117 – 126. doi:10.1016/j.automatica.2005.08.007.

Sloth, C. and Wisniewski, R. Verification of continuous dynamical systems by timed automata. *Formal Methods in System Design*, 2011. 39(1):47–82. doi:10.1007/s10703-011-0118-0.

Tiwari, A. Abstractions for hybrid systems. *Formal Methods in System Design*, 2008. 32(1):57–83. doi:10.1007/s10703-007-0044-3.

Tu, L. W. *An Introduction to Manifolds*. Springer, 2008. doi:10.1007/978-1-4419-7400-6.