Automating Abstraction Computation of Hybrid Systems

Hadi Zaatiti, Jean-Pierre Gallois
Laboratory of Model Driven
Engineering for Embedded Systems
CEA, LIST
Gif-sur-Yvette, 91191, France
firstname.lastname@cea.fr

Lina Ye, Philippe Dague
LRI, Univ. Paris-Sud & CNRS
Univ. Paris-Saclay
Orsay, 91405, France
firstname.lastname@lri.fr

Abstract

Hybrid systems exhibit an interaction of discrete control decisions with continuous physical processes and are present at the core of cyber-physical systems. The verification task of such systems is challenging. In this paper, we are concerned with abstraction methods of hybrid systems. An abstraction can be used to automate the verification of properties such as safety or more complex ones when combined with a model checking algorithm. A tool that automatically computes an abstraction of a given hybrid system with at most polynomial expressiveness is presented. The computed abstraction can be manually refined to achieve further precision guided by the system designer. The tool is tested over several examples.

1 Introduction

Hybrid systems are at the core of many applications mixing discrete control decisions with the behavior of the environment, often modeled as continuous processes. Their verification is a challenging task. It is known that computing the set of reachable states (i.e., the exact set of possible behaviors) of a given hybrid automaton is undecidable. We are concerned with abstraction methods of hybrid systems that retain required information to be later used for decidable property verification at the abstract level (e.g., model checking for safety), giving thus backward partial answer about the status of the property for the hybrid system itself (such as satisfied, not satisfied or does not know).

Abstraction techniques applied to hybrid systems are not new and different methods for abstracting a hybrid system have been studied in the literature. For instance, some tools compute the particular case of abstraction called flow-pipe which is based on performing set-based integrations. Flow-pipe abstractions compute an over-approximating envelope of the reachable set of states starting from a given initial set. These tools are suitable for time-bounded verification (such as bounded model checking) as error propagates and increases rapidly during the computation, the larger the initial set is [FLGD+11, CÁS13].

However, to the best of our knowledge, less tools compute abstractions of hybrid systems using partitioning of the state space or by abstracting transitions of the continuous behavior [Tiw08, Tiw12]. Such abstractions provide coverage with respect to the initial state and are partly compositionally computed. In previous work we showed how the abstractions can be used to verify complex properties of the hybrid system such as diagnosability

In: O. Hasan, S. Tahar, U. Siddique (eds.): Proceedings of the Workshop Formal Verification of Physical Systems (FVPS), Hagenberg, Austria, 17-Aug-2018, published at http://ceur-ws.org

which supposes that the system is partly observable. If a system is diagnosable then it is able to identify the occurrence of a modeled fault using the limited observations [ZYD+18, ZYDG18]. Abstractions of hybrid systems have many use cases such as but not limited to:

- Verification and automated proving: incorporate abstraction computation for the purpose of automated verification in satisfiability modulo theory solvers (SMT) for hybrid systems, user-guided theorem proving, model checking, and invariant synthesis techniques.
- Design, simulation and specification requirements: provide the system designer or architect with verified simulation results from the requirements specification, piloting numerical simulations and unitary tests.
- Provide researchers with insights when modeling complex behaviors.

Related work: The work presented in this paper is inspired from predicate abstraction of hybrid automata introduced in [Tiw08]. The main differences is the instantiation of the abstraction computation to polynomial hybrid systems and predicates expressed as semi-algebraic sets while providing experimental results and abstraction computation time.

[SW13] addressed computing reachability for non-linear systems with polynomial dynamics. Algorithms are proposed to generate a partitioning of the state space, forming an abstraction that is always sound and in some cases complete. The latter is formulated as an optimization problem, which yields results similar to those obtained by invariant synthesis techniques. The termination of the optimization procedure is generally not guaranteed. The difference with the author's work is that we propose an algorithm to compute the time bounds used in the abstraction, however no experimental results are provided for this part.

In this paper, a method for generating abstractions of hybrid systems is presented. Qualitative modeling and reasoning are used to compute the abstraction. The contribution of this paper is the development of a tool that computes abstractions of hybrid automata with polynomial expressiveness in terms of the dynamics model. The abstraction computation are evaluated on different practical examples. A refinement operation is also defined and discussed, allowing the user to reach a further precision. The refinement operation will recompute only required information to keep the abstraction sound. Lastly, an abstraction capturing time constraints is proposed which allows one to handle time constraints that can be abstracted at a qualitative level from the hybrid system through the previously defined abstraction.

The paper is organized as follows. In Section 2 a formal framework for hybrid automata is introduced, then an abstraction of a given hybrid system based on qualitative reasoning is defined. Afterwards, the main algorithms used in the implemented tool for automatically computing the defined abstraction are presented. The abstraction computation is then illustrated on different practical examples and performances of the computations are evaluated. In the last section, a refinement of the abstraction allowing to achieve a higher precision is discussed. This operation recomputes only needed information to keep the abstraction sound. We also explain how to specialize the produced abstraction as a timed one and propose an algorithm to compute it based on flow-pipe construction.

2 Hybrid dynamical systems

Hybrid automata are a mean to model hybrid systems, where each state is twofold with a discrete and a continuous part [Hen96]. The discrete part ranges over a finite domain while the continuous part ranges over the Euclidean space $\mathbb{R}^n$.

Definition 1 (Hybrid automaton)**.** *An n-dimensional hybrid automaton (HA) is $H = (Q, X, S_0, \Sigma, F, Inv, \delta, G, R)$, where:*
• *Q is a finite set of modes (or locations).*
• *X is a set of n real-valued variables whose valuations set is $\boldsymbol{X} \subseteq \mathbb{R}^n$, $S = Q \times \boldsymbol{X}$ is the state space of H, whose elements, noted $(q, \boldsymbol{x})$, are called states.*
• *$S_0 \subseteq S$ is the set of initial states, from which are defined $Q_0 = \{q \mid \exists \boldsymbol{x}\ (q, \boldsymbol{x}) \in S_0\}$, the set of initial modes, and $Init(q) = \{\boldsymbol{x} \mid (q, \boldsymbol{x}) \in S_0\}$ the set of initial continuous values in any initial mode q.*
• *Σ is a finite set of events.*
• *$F : S \to 2^{\mathbb{R}^n}$ is a mapping assigning to each state $(q, \boldsymbol{x}) \in S$ a set $F(q, \boldsymbol{x}) \subseteq \mathbb{R}^n$ (the flow) constraining the time derivative $\dot{\boldsymbol{x}}$ of the continuous part of the mode q by $\dot{\boldsymbol{x}} \in F(q, \boldsymbol{x})$.*
• *$Inv : Q \to 2^{\boldsymbol{X}}$ assigns to each mode q an invariant set $Inv(q) \subseteq \boldsymbol{X}$, constraining the values of continuous*

variables in q: $\forall q \in Q$, $\{\boldsymbol{x} \mid (q, \boldsymbol{x}) \in S\} \subseteq Inv(q)$.
• $\delta \subseteq Q \times \Sigma \times Q$ *is a set of discrete transitions.*
• $G : \delta \to 2^{\boldsymbol{X}}$ *assigns to each transition* $\tau = (q, \sigma, q') \in \delta$ *a nonempty guard set* $G(\tau) \subseteq \boldsymbol{X}$ *such that* $G(\tau) \subseteq Inv(q)$.
• $R(\tau) : G(\tau) \to 2^{Inv(q')}$ *assigns to each guard value new variables reset values after triggering* $\tau = (q, \sigma, q')$.

One can also adopt a relational-based representation and use predicates instead of subsets. Then $F(q)(\mathbf{x}, \dot{\mathbf{x}})$, $Inv(q)(\mathbf{x})$, $G(\tau)(\mathbf{x})$ and $R(\tau)(\mathbf{x}, \mathbf{x}')$ being true means $\dot{\mathbf{x}} \in F(q, \mathbf{x})$, $\mathbf{x} \in Inv(q)$, $\mathbf{x} \in G(\tau)$ and $\mathbf{x}' \in R(\tau)(\mathbf{x})$ respectively. For the rest of the paper, the following is assumed:

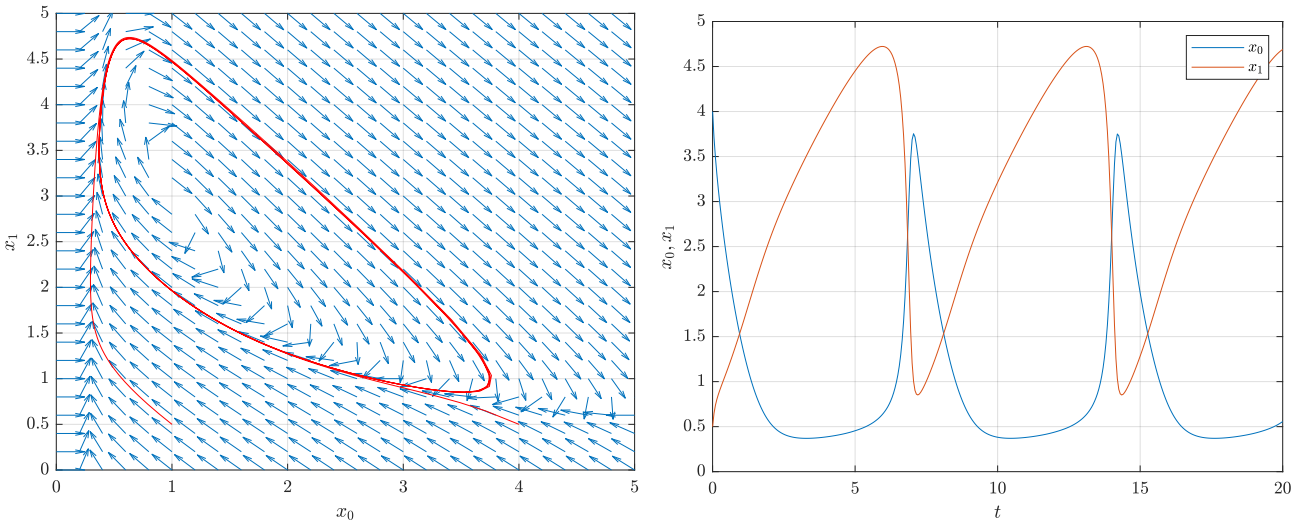
Assumption 1. *Guards in any mode q will be assumed non-intersecting:* $\forall q \in Q, \forall \tau_1 = (q, \sigma_1, q_1) \in \delta, \forall \tau_2 = (q, \sigma_2, q_2) \in \delta, (\tau_1 \neq \tau_2 \Rightarrow G(\tau_1) \cap G(\tau_2) = \emptyset)$.

We now introduce two practical examples of a continuous and a hybrid system that will be adopted throughout the paper.

Example 1 (Continuous System)**.** *The brusselator is a mathematical model used for representing chemical reactions with cyclic change of color. The dynamics are nonlinear. The input model holds a single mode q with singleton flow F given by the variables derivatives as:*

$$\dot{x_0} = 1 - 4x_0 + x_0^2 x_1$$
$$\dot{x_1} = 3x_0 - x_0^2 x_1$$

For illustrative purposes, a number of numerical simulations of the brusselator showing a clockwise rotation (in the (x_0, x_1) plane) of the trajectories in the studied region are performed and observed in Figure 1.



(a) (b)

Figure 1: Brusselator normalized phase plane with numerical simulations

Example 2 (Hybrid System)**.** *Figure 2 illustrates a simple autonomous temperature regulating system (or thermostat) modeled as a hybrid automaton with two modes* on *and* off*, whose switching is witnessed by events B_{on} and B_{off}. The continuous behaviors of each mode are illustrated in Figure 3 with some numerical simulations from randomly generated initial values.*

The language of a hybrid automaton are defined by the set of its trajectories (or semantics) defined below.

Definition 2 (HA semantics)**.** *The semantics of a HA H (also called concrete behavior), denoted by $[[H]]$, is the set of all executions, which are labeled by $L = \Sigma \cup \mathbb{R}_+$:* $(q_0, \boldsymbol{x}_0) \xrightarrow{l_0} (q_1, \boldsymbol{x}_1) ... (q_i, \boldsymbol{x}_i) \xrightarrow{l_i} ...$ *such that* $(q_0, \boldsymbol{x}_0) \in S_0$ *and,* $\forall i$, $(q_i, \boldsymbol{x}_i) \xrightarrow{l_i} (q_{i+1}, \boldsymbol{x}_{i+1})$*, one of the following is true:*

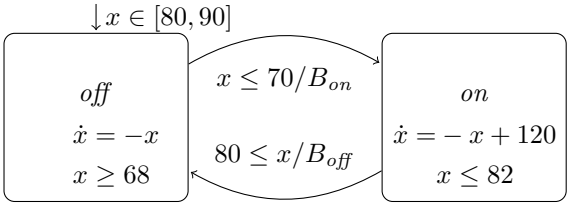


Figure 2: Hybrid automaton modeling a thermostat

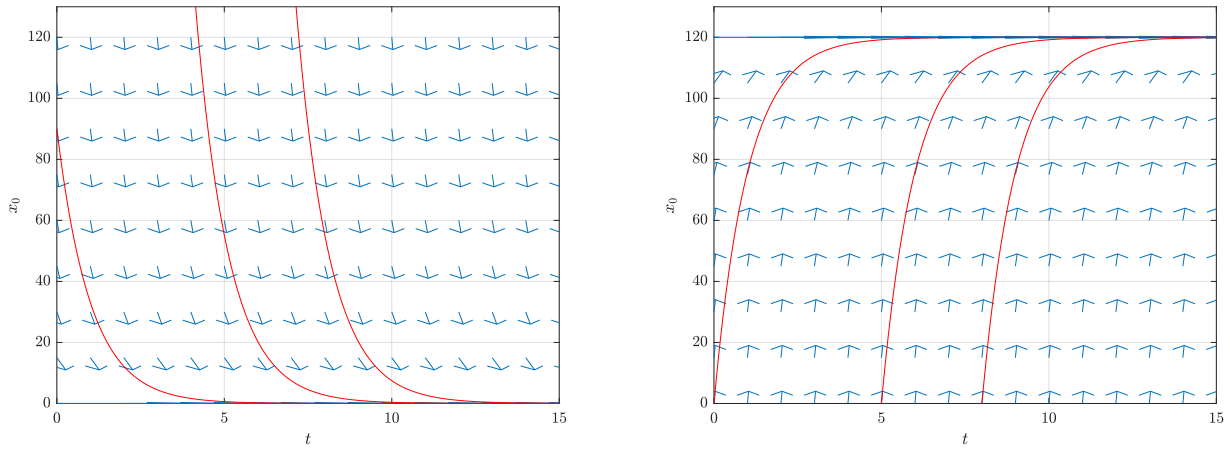


Figure 3: Thermostat phase plane with randomly generated numerical simulations: (a) Mode *off*, (b) Mode *on*

- $l_i = \sigma_i \in \Sigma$, $\tau = (q_i, \sigma_i, q_{i+1}) \in \delta$, $\boldsymbol{x}_i \in G(\tau)$ *and* $\boldsymbol{x}_{i+1} \in R(\tau)(\boldsymbol{x}_i)$*;*
- $l_i = d_i \in \mathbb{R}_+$, $q_i = q_{i+1}$, $\boldsymbol{x}_i, \boldsymbol{x}_{i+1} \in Inv(q_i)$ *and* $\exists x : [0, d_i] \to \boldsymbol{X}$ *continuously differentiable function, with* $x(0) = \boldsymbol{x}_i$, $x(d_i) = \boldsymbol{x}_{i+1}$ *and* $\forall t \in (0, d_i)$ $\dot{x}(t) \in F(q_i, x(t))$ *and* $x(t) \in Inv(q_i)$.

The trace of an execution h, i.e., the sequence of its labels, is a word from $L^\star$ (or L^ω for infinite h), denoted as $trace(h)$. We denote the total time duration of h by $time(h) \in \mathbb{R}_+ \cup \{+\infty\}$, which is calculated as the sum of all time periods in the trace of h: $time(h) = \sum d_i$. The part of execution $h = (off, 80) \xrightarrow{0.15} (off, 69) \xrightarrow{B_{on}} (on, 69) \xrightarrow{0.5} (on, 81) \xrightarrow{B_{off}} (off, 81)...$ is valid for the thermostat example, thus $h \in [[H]]$. Let $\overline{S} = \bigcup_{q \in Q}(\{q\} \times Inv(q)) \subseteq S$ the (infinite) set of invariant satisfying states of H, $\overline{S}_0 \subseteq S_0$ the subset of invariant satisfying initial states and $\to \subseteq \overline{S} \times L \times \overline{S}$ the transition relation defined by Definition 2. The semantics of H is actually given by the labeled transition system $S^t_H = (\overline{S}, \overline{S}_0, L, \to)$, i.e., $[[H]]$ is the set of all paths of S^t_H issued from an initial state. S^t_H is thus a discretization of H with infinite sets of states and of transition labels. It just abstracts continuous flows by timed transitions retaining only information about the source, the target and the duration of each flow and constitutes the finest (timed) abstraction of H we will consider. The timeless abstraction of S^t_H is obtained by ignoring also the duration of flows and thus defined as $S_H = (\overline{S}, \overline{S}_0, \Sigma \cup \{\epsilon\}, \to)$, obtained from S^t_H by replacing any timed transition $(q_i, \mathbf{x}_i) \xrightarrow{d_i} (q_{i+1}, \mathbf{x}_{i+1})$ with $d_i \in \mathbb{R}_+$ by the ϵ transition $(q_i, \mathbf{x}_i) \xrightarrow{\epsilon} (q_{i+1}, \mathbf{x}_{i+1})$, that can be considered as a silent transition. It has infinite set of states but finite set of transitions labels. It constitutes the finest timeless abstraction of H we will consider.

Theorem 1 (Correctness and completeness of the semantics)**.** *Any concrete behavior of H is timed abstracted into an $\overline{S}_0$ rooted path in S^t_H. Conversely, any path in S^t_H that alternates continuous and discrete transitions (in particular any single transition) abstracts a part of a concrete behavior of H and, if F is a singleton function (i.e., deterministic derivative), any $\overline{S}_0$ rooted path in S^t_H abstracts a concrete behavior of H. In this latter case, there is thus no spurious abstract behavior in S^t_H, which expresses faithfully the behavior of H.*

3 Hybrid automata abstraction

We now formally define a hybrid automata abstraction. The abstraction is obtained by partitioning the state space according to the dynamics of each mode of the hybrid system, using qualitative principles and taking into account the guard and reset conditions.

Definition 3 (Continuous space partition)**.** *A (finite) partition P of the space $\mathbb{R}^n$ is a finite set of nonempty connected subsets of $\mathbb{R}^n$ such that every point $x \in \mathbb{R}^n$ is in one and only one of those subsets. We can write $\mathbb{R}^n = \biguplus_{p \in P} p$. An element $p \in P$ is referred to as a partition element and called a region. For a subset E of $\mathbb{R}^n$, we denote by $P(E)$ the subset of regions of P with a nonempty intersection with E.*

The smoothness hypothesis we impose over a partition w.r.t. a given continuous dynamics is that any (finite) path solution of the dynamics crosses only a finite number of times each region, more precisely, $\forall x : [0,1] \to \mathbb{R}^n$ a continuously differentiable function satisfying the flow condition, $\forall p \in P$ a region, $x^{-1}(p)$ is a finite union of intervals. For a HA, it is practical to allow different partitions in different modes.

Definition 4 (Hybrid state space decomposition)**.** *Given a HA H and a set $\mathfrak{P}$ of partitions of $\boldsymbol{X} \subseteq \mathbb{R}^n$, we say that $\mathfrak{P}$ decomposes H if there is a surjective function $d : Q \to \mathfrak{P}$ which associates to each $q \in Q$ a partition $d(q) \in \mathfrak{P}$.*

The continuous space partition relative to a mode contains the initial and invariant sets and the guards satisfiability domains towards other modes and variables reset domains from incoming modes. For $q \in Q$ and $\tau = (q, \sigma, q') \in \delta$, we denote the regions families $d(q)(Init(q))$, $d(q)(Inv(q))$, $d(q)(G(\tau))$ by $d_{Init}(q)$, $d_{Inv}(q)$, $d_G(q,\tau) \subseteq d(q)$ and, for a region $p \in d_G(q,\tau)$, we denote $d(q')(R(\tau)(p \cap G(\tau)))$ by $d_R(q', \tau, p) \subseteq d(q')$. When possible, we try to define d such that $Init(q)$, $Inv(q)$, $G(\tau)$ and $R(\tau)(p)$ are the unions of the regions in those families (if not, those regions families over-approximate them).

Definition 5 (Adjacent regions)**.** *Two distinct regions p_1, p_2 of a partition P of $\mathbb{R}^n$ are adjacent if one intersects the boundary of the other: $p_1 \cap \overline{p_2} \neq \emptyset$ or $\overline{p_1} \cap p_2 \neq \emptyset$, where $\overline{p}$ refers to the closure of p.*

Definition 6 (Decomposition-based timeless abstract automaton of a hybrid automaton)**.** *Given a hybrid automaton $H = (Q, X, S_0, \Sigma, F, Inv, \delta, G, R)$, and a decomposition $(\mathfrak{P}, d)$ of H, we define the timeless abstract (finite) automaton of H with respect to $\mathfrak{P}$ as $DH_{\mathfrak{P}} = (Q_{DH}, Q_{0_{DH}}, \Sigma_{DH}, \delta_{DH})$ with:*

- $Q_{DH} = \{(q,p) | q \in Q, p \in d(q)\}$.
- $Q_{0_{DH}} = \{(q, p_{Init}) | q \in Q_0, p_{Init} \in d_{Init}(q)\}$.
- $\Sigma_{DH} = \Sigma \cup \{\epsilon\}$.
- $((q_i, p_k), \sigma, (q_j, p_l)) \in \delta_{DH}$ *iff one of the two following conditions is true:*
  - $\sigma \in \Sigma$ *and* $p_k \in d_G(q_i, \tau)$ *and* $p_l \in d_R(q_j, \tau, p_k)$ *where* $\tau = (q_i, \sigma, q_j) \in \delta$.
  - $q_i = q_j$ *and* $\sigma = \epsilon$ *and* $p_k, p_l \in d_{Inv}(q_i)$ *are adjacent regions and* $\exists d \in \mathbb{R}_+^*$ *and* $\exists x : [0,d] \to \boldsymbol{X}$ *continuously differentiable function such that* $\forall t \in (0,d)\ \dot{x}(t) \in F(q_i, x(t))$, $\forall t \in [0,d]\ x(t) \in Inv(q_i)$, $x(0) \in p_k$, $x(d) \in p_l$, $\exists c\ 0 \leq c \leq d\ \forall t \in (0,c)\ x(t) \in p_k\ \forall t \in (c,d)\ x(t) \in p_l$ *and* $x(c) \in p_k \cup p_l$.

The defined timeless abstract automaton encodes reachability with adjacent regions of the state space, the events in Σ witnessing mode changes and ϵ transitions representing a continuous evolution between adjacent regions in the same mode. Notice that $((q_i, p_k), \sigma, (q_j, p_l)) \in \delta_{DH} \Rightarrow \exists \mathbf{x}_k \in p_k\ \exists \mathbf{x}_l \in p_l\ (q_i, \mathbf{x}_k) \xrightarrow{\sigma} (q_j, \mathbf{x}_l)$ in S_H, the converse being true for $\sigma \in \Sigma$. The mapping $\alpha_{\mathfrak{P}}$ defined by $\alpha_{\mathfrak{P}}((q, \mathbf{x})) = (q, p)$ with $p \in d(q)$ and $\mathbf{x} \in p$ defines a surjective timeless abstraction function $\alpha_{\mathfrak{P}} : \overline{S} \to Q_{DH}$. If the flow condition F is a singleton, $\alpha_{\mathfrak{P}}$ maps any transition of S_H to a unique path in $DH_{\mathfrak{P}}$. The coarsest timeless abstract automaton is obtained when partitions of $\mathfrak{P}$ have all a unique region $p = \mathbf{X}$ and is thus (Q, Q_0, Σ, δ), i.e., the discrete part of H without its continuous part. It corresponds to the coarsest timeless abstraction function $\alpha_{\{\{\mathbf{X}\}\}}((q, \mathbf{x})) = q$. For our previous thermostat example, this gives $(\{off, on\}, \{off\}, \{B_{on}, B_{off}\}, \{(off, B_{on}, on), (on, B_{off}, off)\})$ and the abstraction of the execution h given previously in Section 2 is just $off \xrightarrow{B_{on}} on \xrightarrow{B_{off}} off ...$

Theorem 2 (Timeless abstraction completeness)**.** *Given a decomposition $\mathfrak{P}$ of H, any concrete behavior of H is timeless abstracted into a $Q_{0_{DH}}$ rooted path in $DH_{\mathfrak{P}}$ and any transition of $DH_{\mathfrak{P}}$ abstracts a part of a concrete behavior of H. If the flow condition F is a singleton function then the timeless abstraction function $\alpha_{\mathfrak{P}}$ defines a trace preserving mapping (still denoted by $\alpha_{\mathfrak{P}}$) from $\overline{S}_0$ rooted paths in S_H (i.e., timeless executions of H) to $Q_{0_{DH}}$ rooted paths in $DH_{\mathfrak{P}}$ and thus the language defined by S_H is included in the language defined by $DH_{\mathfrak{P}}$.*

Obviously, a path in $DH_{\mathfrak{P}}$ does not necessarily abstract a concrete behavior of H (as the behaviors parts abstracted by the individual transitions may not connect) which expresses that abstraction creates spurious behaviors.

4 Algorithmic computation of the abstraction

In this section we introduce a tool that automatically computes the previously defined abstraction of a given hybrid system. The characterization and computation of abstract states and transitions is implemented in *C++*. The tool requires the following freely available libraries and solvers:

- The *Boost* libraries, mainly for parsing the hybrid automaton input model and generating graphical and textual outputs.
- *Nlopt*, a mathematical optimization library for non linear functions.
- *Qepcad*, for quantifier elimination using cylindrical algebraic decomposition (CAD) solver.
- *Matplotlib* and *Graphviz* for visualization.

In the previous section, we formally defined an abstraction $DH_{\mathfrak{P}}$ of a given hybrid system H according to a partition $\mathfrak{P}$. We now show the algorithms to automatically compute it. The computation can be done on the hybrid system modes separately, then a linking operation is performed between the mode abstractions. The main steps are summarized as follows:

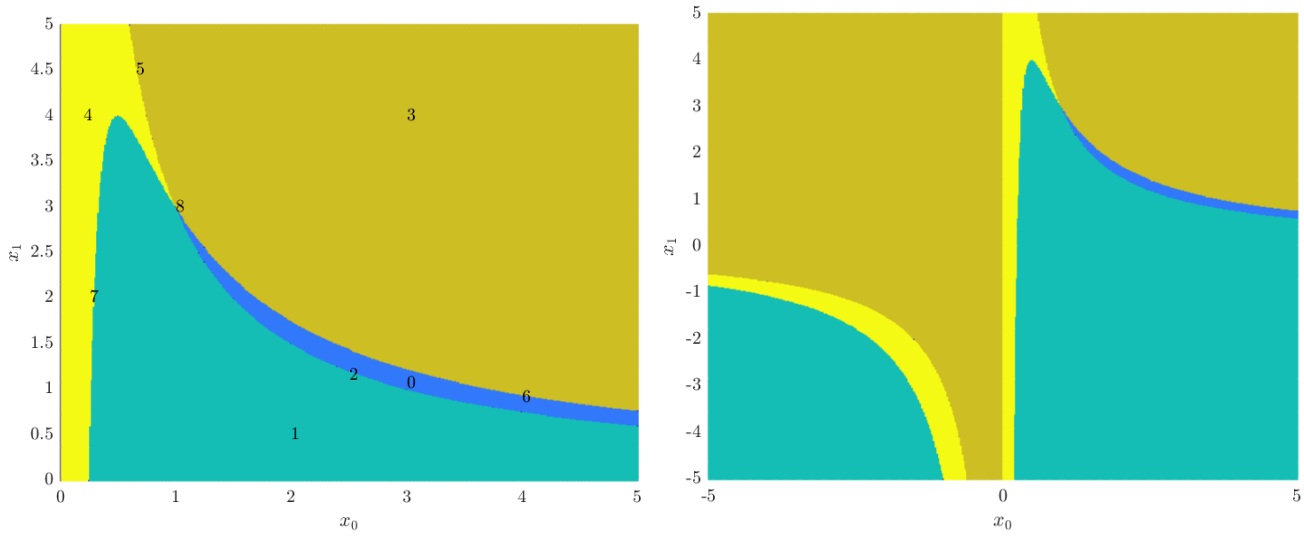
- For each mode q, compute symbolic representation of abstract states.
- For each computed symbolic state st, evaluate the presence of a transition towards abstract states whose concretization is adjacent to the concretization of st.
- Apply the linkage operation.

4.1 Symbolic computation of the abstract states

Partitioning rules. Consider a mode q of H. We will apply the single mode abstraction algorithm to compute an abstraction $DH_{\{d(q)\}}$ of q. In the existing literature, partitions are chosen enough regular and smooth, with regions in any dimension from 1 to n such as (from simpler to more complex) rectangles, zonotopes, polytopes and Taylor models. The choice is often guided by the dynamics and the property one wishes to verify. For example, consider a continuous system with dynamics F. A way to obtain an abstract reachability mapping would be to identify regions of the state space that preserve the sign vector of F, i.e., the derivative signs for each dimension would keep the same value (either negative or positive or null) in the same region. Thus, the regions would be the connected components of the 3^n subsets E_s of $\mathbb{R}^n$ parametrized by sign vectors $s \in \{-1, 0, +1\}^n$: $E_s = \{\mathbf{x} \in \mathbf{X} \mid \forall i, 1 \leq i \leq n,\ \dot{\mathbf{x}}^i < 0 \text{ if } s^i = -1,\ \dot{\mathbf{x}}^i = 0 \text{ if } s^i = 0,\ \dot{\mathbf{x}}^i > 0 \text{ if } s^i = +1\}$. Note that in the case of regions expressed as polynomial inequalities, the sets conserving the sign vector are not necessarily connected, in such case we will reason over non connected sets and use invariants ($Inv(q)$) to separate them.

Expressing abstract states. We use semi-algebraic sets to represent regions using (unions of) (in)equalities of polynomials over X. For example, if X contains two variables, we can express a circular set as: $(x - x_0)^2 + (y - y_0)^2 < r^2$ with x_0, y_0 and r as given rational numbers. Now consider our previous brusselator example. By applying the rules of preservation of the sign vector of F, we obtain 9 abstract states illustrated as colored regions and their intersection boundaries (Figure 4.a). Notice that some generated abstract states could represent non convex sets (region 4 in Figure4a) non-connected sets (Figure 4b). To further partition algorithmically the state-space into connected sets we refer the reader to algorithms in [BPR98].

Computing the abstract states. This computation is implemented in the following way. For a mode q of the hybrid automaton, a stack $Stack(q)$ is initialized with polynomials representing the initial and guard sets, incoming resets and dynamics. The stack is then extended as follows: each predicate from the stack is copied three times and copies are assigned $+$, $-$ and *null* symbols. The regions expressions are obtained by symbolically computing the cross product of every element in the stack with all the other elements that have a different predicate. The first computation step is symbolic, thus, some regions are not feasible (i.e., are empty sets)



(a) (b)

Figure 4: Brusselator partition according to null-clines and equilibrium point
(a) $Inv(q) = \{x_0 > 0 \wedge x_1 > 0\}$, each abstract state is a connected set (b) $Inv(q) = true$, some abstract states are not connected sets

example. A first check is performed to eliminate all empty regions. This is implemented by finding whether or not there exists a solution to the semi-algebraic set representing the region. The evaluation is performed using a quantifier elimination solver applying cylindrical algebraic decomposition and will be discussed in the next section. Abstract states whose region representation admits no solution are simply discarded from the stack.

4.2 Computation of the transitions between abstract states

Expressing a transition. Consider an abstract state st whose concrete region is expressed as a semi-algebraic set R. For each region adjacent to R, we evaluate whether some trajectories in R are to cross in bounded time to the considered neighboring region. For our previous brusselator example, consider the two regions:

$$R_1 : 1 - 4x_0 + x_0^2 x_1 > 0 \wedge 3x_0 - x_0^2 x_1 > 0 \tag{1}$$
$$R_2 : 1 - 4x_0 + x_0^2 x_1 > 0 \wedge 3x_0 - x_0^2 x_1 < 0 \tag{2}$$

Each one is adjacent to the region B which is their common boundary:

$$B : 1 - 4x_0 + x_0^2 x_1 > 0 \wedge 3x_0 - x_0^2 x_1 = 0 \tag{3}$$

In a general setting, we express two open regions R_1 and R_2 adjacent to their common boundary B (of dimension $n-1$) as $R_1 : p > 0$, $R_2 : p < 0$ and $B : p = 0$ where p is a polynomial (the reasoning would be the same with for example R_2 closed, thus $B \subseteq R_2$, expressing R_2 as $p \leq 0$ and checking for transition from R_2 to R_1). We study the orientation of the vector field F in B. Let $\vec{n} = \frac{\partial p}{\partial x}$ be the normal vector of p, it points towards $p > 0$. The presence of a transition in the abstraction $DH_{\mathfrak{P}}$ of H is computed by evaluating the sign of the projection of F on $\vec{n}$ expressed by the scalar product $\vec{n}.F|_{\mathbf{x}}$ on the boundary: $\exists \mathbf{x} \in \mathbf{X}, (p(\mathbf{x}) = 0) \wedge (\vec{n}.F|_{\mathbf{x}} > 0)$. Figure 5 illustates the case of a positive scalar product of the flow vector F with the normal vector over some boundary given by $p = 0$.

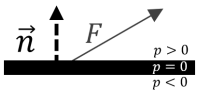


Figure 5: Computing transitions by evaluating the projection of the vector field on the normal vector

For the previously considered regions in equations (1,2) the presence of a trajectory from R_2 to B and from B to R_1 is expressed as the truth value of the following formula :

$$\exists \mathbf{x} \in \mathbf{X}, (3x_0 - x_0^2 x_1 = 0) \wedge (1 - 4x_0 + x_0^2 x_1 > 0) \wedge (3 - 12x_0 - 2x_0 x_1 + 11x_0^2 x_1 - 3x_0^3 - 2x_0^3 x_1^2 + x_0^4 x_1) > 0$$

If the formula evaluates to *true* then a transition is added from the abstract states of B to R_1 (and also from R_2 to B). The analog result holds obviously by replacing $\vec{n}.F|_{\mathbf{x}} > 0$ by $\vec{n}.F|_{\mathbf{x}} < 0$ and exchanging R_1 and R_2. The procedure is repeated for every abstract state until all transitions are computed. Note that, to be semantically correct, every mode invariant $Inv(q)$ must be explicitly added (i.e., with a boolean $\wedge$ operator) to the transition evaluation formula.

Computing the transitions. We perform quantifier elimination using *CAD* over polynomials to evaluate the *true* or *false* value of each transition computed formula. Our tool is interfaced with *Qepcad* that implements this method [Bro03].

4.3 Linking the single modes abstractions

To link the modes abstractions together, we express each region belonging to the guard condition for a jump from a mode q_1 to a mode q_2 in terms of the partition of mode q_2. In other words, if there is no reset, we compute $\alpha_{d(q_2)}(G)$, given $\alpha_{d(q_1)}(G)$. This is implemented by checking the satisfiability of the following constraint for each region $R \in d(q_2)$:

$$\exists \mathbf{x} \in \mathbf{X}, \alpha_{d(q_1)}(G) \wedge \alpha_{d(q_2)}(R)) \tag{4}$$

If satisfied, a mode change transition is thus added from region G to R.

5 Experimental results

In this section, experimental results are provided from several examples of continuous and hybrid systems. The presented abstractions are automatically generated using the tool. The computations are achieved on a machine equipped with an Intel Core i5-3210M CPU operating at a 2.5 *Ghz* frequency.

Example 1: The brusselator.
We are interested in the behavior of the brusselator in the first quadrant of the (x_0, x_1) plane, thus we add to $Inv(q)$ the constraints: $x_0 > 0, x_1 > 0$ as invariants. The system is two-dimensional, hence the computed abstraction according to flow sign vector contains $3^2 = 9$ states. The tool successfully analyzes the example and the abstraction procedure of the model terminates in 3293 milliseconds. The produced abstraction graph is visible in Figure 6a and the computed transitions in the continuous space are illustrated in Figure 6b.

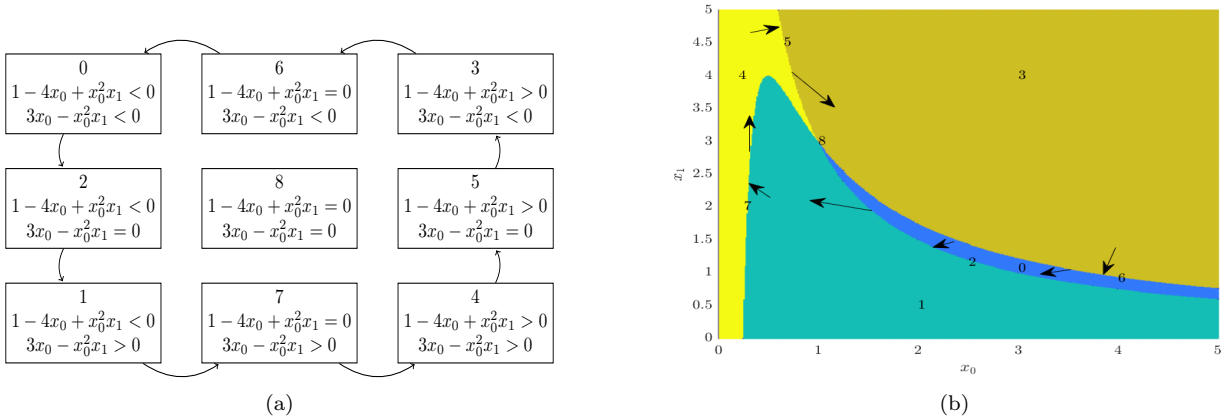


Figure 6: Brusselator computed abstraction
(a) Automata graph view (b) Continuous space view

The qualitative simulation shows what behavior of the system is impossible and what behavior is possible. Using a numerical simulator, it would take a number of simulations to show the cyclic behavior of the brusselator.

However Figure 6 shows that the rotation direction of the brusselator cycle cannot be counter-clockwise in a single abstraction step and without any further refinements and that is valid for any trajectory in the unbounded ($x_0 > 0 \wedge x_1 > 0$) domain. Note that state 8 corresponds to the equilibrium point, i.e., in a neighborhood around state 8 no trajectories are to reach it in any time. As a result, trajectories come neither in nor out, thus no outgoing or incoming transitions are associated to state 8.

Example 2: The thermostat.
The previously introduced thermostat is abstracted using our tool within 13340 milliseconds. The abstraction is illustrated in Figure 7. Green states correspond to guard satisfiability domain and the blue states are regions interpreted from the other mode and purple states are regions produced if both mode invariants were set to *true*.

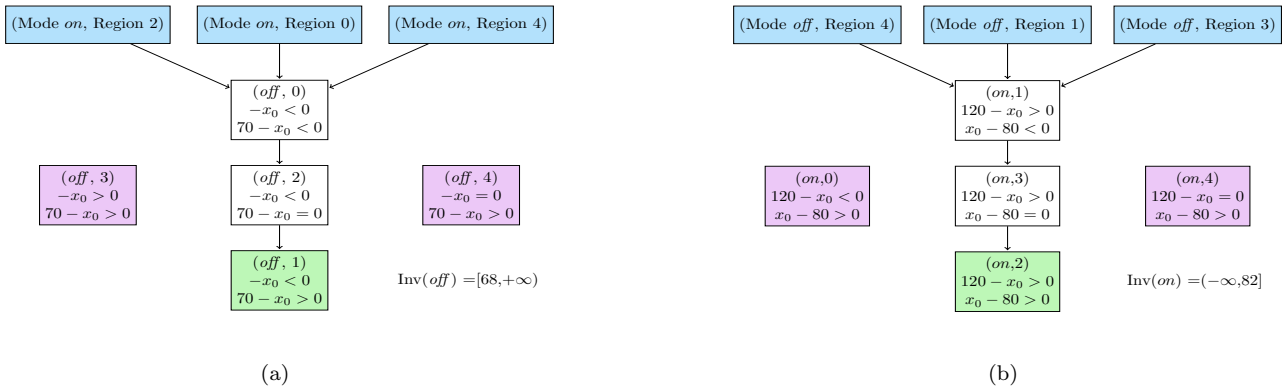


Figure 7: Thermostat qualitative abstraction: (a) Mode *off*, (b) Mode *on*

Observing the abstractions graph we can immediately see that some regions of the state space are time invariants. These regions belong to the mode switching guard condition. The abstract state of the initial set $[80, 90]$ is region 0 in mode *off*. From state 0 an abstract trajectory exists, a mode change is possible to region 1 in mode *on* and going back to mode *off* is possible. The abstraction is sufficient to provide a proof that regions 2 and 3 are to be crossed by the system for a mode change to happen. The arrows from the blue regions are transitions computed from the linking step that was presented in the previous section.

An aspect of the produced abstraction is that it can be applied to different cases of initial sets without recomputing the whole abstraction. To observe abstract trajectories from two given initial sets $Init$ and $Init'$, it is only required to compute $\alpha(Init)$ and $\alpha(Init')$. This aspect provides time efficient verification when studying the hybrid systems behaviors under varying initial sets. If the verification was performed numerically, the executions from each initial set should be computed separately.

Performance evaluation. It is important to note that computing the transitions can be parallelized, once the abstract states are computed for a single mode. For the moment this parallelization is not implemented in the presented tool. Parallelizing the computations will have a drastic impact on the computation time of the abstraction and will allow to handle systems with large number of modes firstly and with higher dimensions (i.e., the number of continuous variables involved).

6 Refining and timing the abstraction

The computed abstraction holds reachability information between adjacent regions, however time is not taken into account. So, for example the abstraction cannot be used to determine in how much time does a transition occur thus it cannot be used to verify temporal properties in a quantitative way. Moreover, if the abstraction is to be used for model checking for example, it would be often necessary to add more information to the abstraction for further precision. For these reasons, in this section, we define and discuss the refinement operations of the previously produced abstraction by taking into account time constraints. For the moment, refinement is implemented in our tool as a user guided operation however the time constraints on the abstraction is not yet implemented.

6.1 Refinement operation

Refining the abstraction consists in adding further information from the hybrid system to obtain a more precise abstraction. The idea is to split one or more regions into several regions by adding further boundaries to the decomposition. This is applied per mode of the hybrid system. What is important to notice is that many of the previously computed transitions remain valid after a refinement operation. We would only need to algorithmically check for the transitions that require to be recomputed and reevaluated.

The following are the algorithmic steps to perform for a refinement operation. Consider a new boundary region B_{new} (typically of dimension $n-1$) to be added to the abstraction. In order to keep the abstraction sound, the following needs to be performed:

1. Find the set of regions $\{R_{compute}\}$ from the previous decomposition that intersect B_{new}.
2. For the set of regions $\{R_{old}\}$ that do not intersect B_{new}, leave their incoming and outgoing transitions intact. Re-express these regions as $R_{old} := R_{old} \wedge p_{B_{new}}$ where $p_{B_{new}}$ is the half space delimited by B_{new} containing R_{old}.
3. Process each region $R_{compute}$ intersecting B_{new}.
   - Find which boundaries (in all dimensions) of $R_{compute}$ intersect B_{new}.
   - Compute the newly obtained regions.
   - Recompute the in and out transitions relative to those new regions as in Section 4.2.

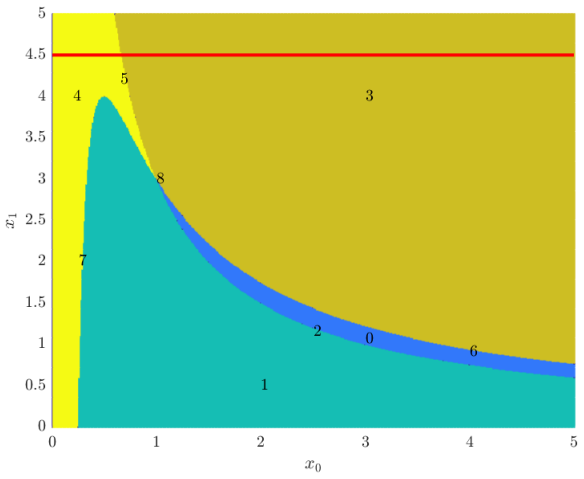


Figure 8: Adding a refinement region to a previous decomposition

Example: Consider adding a new boundary region expressed by $B_{new} : x_1 = 4.5$ to the previous decomposition of the brusselator. We illustrate this region in red on Figure 8. By simple observation, we can see that some transitions such as $0 \to 2$, $2 \to 1$ and $7 \to 4$ remain valid in the newly considered decomposition. By applying the previous algorithm, $R_{compute}$ will contain the regions $4, 5$ and 3.

6.2 Adding time

A suitable representation for systems with time constraints is a particular class of hybrid systems called timed automata. In this section we introduce timed automata, propose and discuss an extension of the previous abstraction that adds time bounds associated to regions allowing for a larger class of properties to be verified using the timed abstraction.

Timed automata are a sub-class of hybrid automata where the continuous variables, called clocks, have all first order derivatives equal to one, i.e., time elapses identically for all. $C(X)$ denotes the set of constraints over a clocks set X: either primitive constraints of the form $x_i \ op \ c_i$, where $c_i \in \mathbb{R}_+$ and $op \in \{<, \leq, =, \geq, >\}$, or finite conjunctions of primitive constraints. The satisfiability set of a constraint is thus a rectangle in $\mathbb{R}_+^n$ and we identify $C(X)$ to the set of those rectangles.

Definition 7 (Timed automaton)**.** *A timed automaton (TA) is a hybrid automaton (HA)* $T = (Q, X, S_0, \Sigma, F, Inv, \delta, G, R)$ *such that:*
• $S_0 \subseteq Q \times \{\boldsymbol{0}\}$*, or* $S_0 = Q_0 \times \{\boldsymbol{0}\}$ *with* Q_0 *initial modes set.*

• $\forall q \in Q\ F(q,.) = \mathbf{1}$, *which means that the dynamics of clocks evolution in each mode* q *is given by* $\dot{x}_i = 1$.
• $Inv : Q \to C(X)$ *associates to each mode* q *a rectangle invariant in* $\boldsymbol{X}$. *We require* $\mathbf{0} \in Inv(q_0)$ *for all* $q_0 \in Q_0$.
• $G : \delta \to C(X)$ *associates to each discrete transition* (q, σ, q') *a rectangle guard in* $Inv(q)$.
• $\forall \tau \in \delta\ \exists Y(\tau) \subseteq X\ \forall \boldsymbol{x} \in G(\tau)\ R(\tau)(\boldsymbol{x}) = \{\boldsymbol{x'}\}$ *with* $\boldsymbol{x'_i} = 0$ *if* $x_i \in Y(\tau)$ *and* $\boldsymbol{x'_i} = \boldsymbol{x_i}$ *otherwise, i.e., clocks in* $Y(\tau)$ *are reset to zero with transition* τ, *the others keeping their values.*

The notation of a timed automaton T is generally simplified as $T = (Q, X, Q_0, \Sigma, Inv, \delta, G, Y)$. The semantics $[[T]]$ of T as a HA can be simplified by merging together successive timed transitions between two discrete transitions and summing up their time period labels. An execution is thus a sequence h of alternating time steps (possibly with 0 time period) and discrete steps of the form $(q_0, \mathbf{x}_0) \xrightarrow{d_1} (q_0, \mathbf{x}_0 + d_1) \xrightarrow{\sigma_1} (q_1, \mathbf{x}_1) \xrightarrow{d_2} ...$ whose trace $trace(h)$ is the timed word $d_1\sigma_1 d_2 ... \in \mathbb{R}_+(\Sigma\mathbb{R}_+)^*$ and duration is $time(h) = \sum d_i$.

Encoding reachability and time constraints. We now show how to abstract a HA into a TA that partly captures the reachability and time constraints at the level of the regions by using state space decomposition. To intuitively introduce this section, consider a partition P of the $\mathbb{R}^n$ state space of a continuous system with arbitrary dynamics F, the set of trajectories entering a region $p \in P$ is in one of these two cases:

1. either at least one of the trajectories ends up trapped inside p for all future times
2. all trajectories exit p to an adjacent region within a bounded time under the continuity assumption

In the first case, no time constraint can be associated to the region p unless a reshaping of p is applied (i.e., via a refinement operation); in the latter, it is possible to compute time constraints satisfied by all trajectories entering and leaving the region p.

Definition 8 (Region time interval and time bounds)**.** *Given a continuous system* CS, *a partition* P *of* $\mathbb{R}^n$ *and* $p \in P$ *one of its regions, we say that* $I_p = [t_{min}, t_{max}]$, *with* $t_{min}, t_{max} \in \mathbb{R}_+ \cup \{+\infty\}$, *is a region time interval of* p *for* CS *if all trajectories of the* CS *entering* p *at time* t *leave* p *at time* $t + t_{min}$ *at least and* $t + t_{max}$ *at most.* t_{min} *and* t_{max} *are lower and upper time bounds of* p.

For a hybrid automaton, we denote a time interval relative to the region p in mode q as $I_{(q,p)}$.

Definition 9 (Decomposition-based abstract TA of a HA)**.** *Given a HA* $H = (Q, X, S_0, \Sigma, F, Inv, \delta, G, R)$, *a decomposition* $\mathfrak{P}$ *of* H *and the timeless abstract automaton* $DH_{\mathfrak{P}} = (Q_{DH}, Q_{0_{DH}}, \Sigma_{DH}, \delta_{DH})$ *of* H *with respect to* $\mathfrak{P}$, *we define the abstract TA of* H *with respect to* $\mathfrak{P}$ *as* $TH_{\mathfrak{P}} = (Q_{DH}, \{c\}, Q_{0_{DH}}, \Sigma_{DH}, Inv_{TH}, \delta_{DH}, G_{TH}, Y_{TH})$ *with:*
• $Inv_{TH}((q,p)) = [0, t_{max}]$ *where* $I_{(q,p)} = [t_{min}, t_{max}]$.
• $\forall \tau = ((q,p), \sigma, (q_1, p_1)) \in \delta_{DH}$ *with* $I_{(q,p)} = [t_{min}, t_{max}]$, $G_{TH}(\tau) = [t_{min}, +\infty)$ *if* $\sigma = \epsilon$ *and* p *does not intersect any reset set (i.e.,* $\forall \tau' = (q', \sigma', q) \in \delta_{DH}\ p \notin d(q)(R(\tau')(G(\tau'))))$ *or* $[0, +\infty)$ *else.*
• $\forall \tau \in \delta_{DH}$, $Y_{TH}(\tau) = \{c\}$.

The defined abstract TA inherits the previously computed reachability relations with adjacent regions. The events in Σ witness mode changes and ϵ transitions represent a continuous evolution between adjacent regions in the same mode. Time constraints are added to a state (q,p) for which an interval $I_{(q,p)}$ is computable as non-trivial (i.e., $I_{(q,p)} \neq [0, +\infty)$), by using one local clock c (reset at 0 in each state) that measures the sojourn duration t in each state (q,p), i.e., in each region p, and coding these constraints by means of invariant and guard of c in each state. The invariant codes the maximum sojourn duration as the upper time bound of the region p and the guard codes the minimum sojourn duration as the lower time bound of the region p when both entering and leaving the region are not the result of discrete jumps (controlled here directly for the out-transition and by requiring that p does not intersect any reset set for all possible in-transitions). In the thermostat example, consider the partition into two regions associated to the mode *off* given by the initial state $(off, [80, 90])$ and by $(off, [68, 80))$. Then we take as time bounds for $(off, [80, 90])$ $t_{min} = 0$ and $t_{max} = 0.12$ (the exact upper bound, i.e., the time for the temperature to decrease from 90 to 80 is $Log(\frac{9}{8})$). Thus, we define in the abstract TA $Inv_{TH}((off, [80, 90])) = [0, 0.12]$. A beginning of one execution of the TA is for example $(off, [80, 90]) \xrightarrow{0.08} (off, [68, 80])$. A timed abstraction of the thermostat example is illustrated in Figure 9, the timed automaton keeps the same reachability relations between regions that were previously computed while adding the assigned time bounds. To be noted that we applied a split operation in region 0 of mode *off* to show the initial set $[80, 90]$ and the associated time bounds. ϵ and ϵ' are any strictly positive reals and are used to express the time bounds of a boundary region.

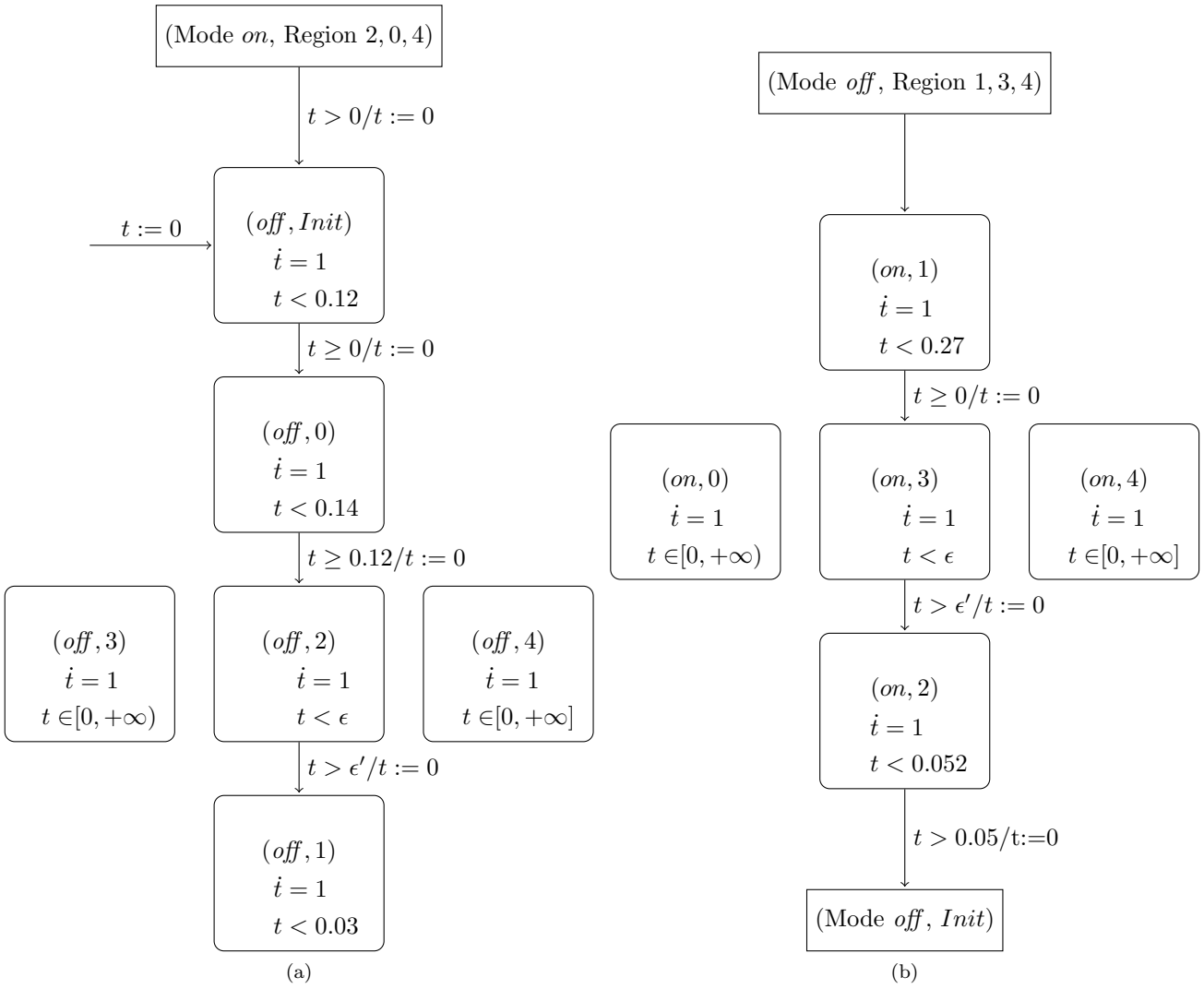


Figure 9: Timed abstraction of the thermostat system mode (a) Mode *on*, (b) Mode *off*

Theorem 3 (Timed abstraction completeness)**.** *Given a decomposition $\mathfrak{P}$ of H, any concrete behavior of H is timed abstracted into an execution in $TH_{\mathfrak{P}}$. If the flow condition F is a singleton function then the abstraction function $\alpha_{\mathfrak{P}}$ defines a mapping, denoted by $\alpha^t_{\mathfrak{P}}$, from $\overline{S}_0$ rooted paths in S^t_H (i.e., executions of H) to executions in $TH_{\mathfrak{P}}$. This mapping is trace preserving once ϵ labels are erased from executions traces in $TH_{\mathfrak{P}}$ and time period labels are added up between two consecutive events labels in both executions traces in S^t_H and in $TH_{\mathfrak{P}}$. This means that, for any execution $(q_0, \boldsymbol{x}_0) \xrightarrow{w}_* (q_i, \boldsymbol{x}_i) \in [[H]]$, with $w \in L^*$ (where $L = \Sigma \cup \mathbb{R}_+$), it exists a unique execution $(q_0, p_0) \xrightarrow{w'}_* (q_j, p_j) \in [[TH_{\mathfrak{P}}]]$, with $w' \in L'^*$ (where $L' = L \cup \{\epsilon\}$), $\boldsymbol{x}_0 \in p_0$, $q_j = q_i$, $\boldsymbol{x}_i \in p_j$, $w'_{|\Sigma} = w_{|\Sigma}$ (where $_{|\Sigma}$ is the projection of timed words on words on Σ^*) and, for any two successive events $w_l = w'_{l'}$ and $w_m = w'_{m'}$ of $w_{|\Sigma}$, $\sum_{l'<k'<m', w'_{k'} \neq \epsilon} w'_{k'} = \sum_{l<k<m} w_k$.*

Computing time bounds. Now we discuss how to compute the time bounds of the hybrid system (Def. 8). Finding the exact minimum and maximum sojourn times is not always applicable. Hence, it is rather more practical to consider upper and lower bounds of these sojourn times. To find a safe upper bound of the maximum sojourn time, the idea is to construct a flow-pipe $Flow$ of total duration $N\Delta t$ where Δt is the time step and $N \in \mathbb{N}$ is to be found. $Flow$ is initialized from a boundary B of a considered region p. $Compute\_Flow(B, \Delta t)$ is a procedure performing set integration initially starting from B and given Δt. We apply Algorithm 1 to obtain t_{max} of a region p.

Flow-pipes received considerable attention in the literature, different tools exist and were successful in analyzing hybrid automata from linear to polynomial dynamics [FLGD$^+$11, CÁS13]. The previous algorithm can

Algorithm 1 Computing Time Bounds of a region p

Input: set S of boundaries of a region p, time horizon T
for all $B_i \in S$ **do**
  $t_{max_i} \leftarrow 0$
  while $t_{max_i} < T$ **do**
    $Flow \leftarrow Compute\_Flow(B_i, \Delta t)$
    $t_{max_i} \leftarrow t_{max_i} + \Delta t$
    if $(Flow \cap p = \emptyset)$ break
  end while
  if $(t_{max_i} \geq T)$ return "max reached"
end for
return maximum(t_{max_i})

be optimized by not considering boundaries for which formula (1) evaluates to $false$. Obviously, unless bounded by T, termination of algorithm 1 is generally not guaranteed. One can check for conditions for which finite time bounds exist and adapt the maximum time horizon T accordingly. For example:

Proposition 1 (Sojourn bounds)**.** *A sufficient but not necessary condition for the region p to have finite time bounds (t_{max} finite, thus real nonnegative constant) is that* $\exists i\ 1 \leq i \leq n \quad \forall \boldsymbol{x} \in \overline{p} \quad \dot{\boldsymbol{x}}_i \neq 0$.

7 Conclusion

In this article, we provided a formal framework for abstracting hybrid automata into discrete-event systems, by decomposing the continuous state space into a finite number of geometric regions, and into timed automata, by adding time constraints approximating safely the sojourn time of trajectories in each region. We implemented the discrete automata abstraction computation into a tool which we tested over several examples to evaluate performances. We proposed a method to refine the abstraction by further decomposing the state space. Our next work is to extend our tool by implementing both the time constraints computation, and thus the timed automata abstraction, and the abstraction refinement process in the framework of CEGAR (Counter-Example Guided Abstraction Refinement), both being presently done in great part manually.

References

[BPR98] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. Complexity of computing semi-algebraic descriptions of the connected components of a semi-algebraic set. In *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation*, ISSAC '98, pages 25–29, New York, NY, USA, 1998. ACM.

[Bro03] Christopher W Brown. QEPCAD B: a program for computing with semi-algebraic sets using CADs. *ACM SIGSAM Bulletin*, 37(4):97–108, 2003.

[CÁS13] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.

[FLGD+11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 379–395. Springer, 2011.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS)*, pages 278–292. IEEE Computer Society Press, 1996.

[SW13] Christoffer Sloth and Rafael Wisniewski. Complete abstractions of dynamical systems by timed automata. *Nonlinear Analysis: Hybrid Systems*, 7(1):80–100, 2013.

[Tiw08] Ashish Tiwari. Abstractions for hybrid systems. *Formal Methods in Systems Design*, 32:57–83, 2008.

[Tiw12] Ashish Tiwari. Hybridsal relational abstracter. In *International Conference on Computer Aided Verification*, pages 725–731. Springer, 2012.

[ZYD+18] Hadi Zaatiti, Lina Ye, Philippe Dague, Jean-Pierre Gallois, and Louise Travé-Massuyès. *Abstractions Refinement for Hybrid Systems Diagnosability Analysis*, pages 279–318. Springer International Publishing, Cham, 2018.

[ZYDG18] Hadi Zaatiti, Lina Ye, Philippe Dague, and Jean-Pierre Gallois. Counterexample-guided abstraction-refinement for hybrid systems diagnosability analysis. In Marina Zanella, Ingo Pill, and Alessandro Cimatti, editors, *28th International Workshop on Principles of Diagnosis (DX'17)*, volume 4 of *Kalpa Publications in Computing*, pages 124–143. EasyChair, 2018.