Automatic Rectangular Refinement of Affine Hybrid Systems

Laurent Doyen[1⋆], Thomas A. Henzinger[2,3 ⋆⋆], and Jean-François Raskin[1]

[1] Université Libre de Bruxelles (ULB) Brussels, Belgium

[2] École Polytechnique Fédérale de Lausanne (EPFL) Lausanne, Switzerland

[3] University of California (UC) Berkeley, U.S.A.

Abstract. We show how to automatically construct and refine rectangular abstractions of systems of linear differential equations. From a hybrid automaton whose dynamics are given by a system of linear differential equations, our method computes automatically a sequence of rectangular hybrid automata that are increasingly precise overapproximations of the original hybrid automaton. We prove an optimality criterion for successive refinements. We also show that this method can take into account a safety property to be verified, refining only relevant parts of the state space. The practicability of the method is illustrated on a benchmark case study.

1 Introduction

Hybrid systems are digital real-time systems embedded in analog environments. A paradigmatic example of a hybrid system is a digital embedded control program for an analog plant environment, like a furnace or an airplane: the controller state moves discretely between control modes, and in each control mode, the plant state evolves continuously according to physical laws. Those systems combine discrete and continuous dynamics. Those aspects have been studied in computer science and in control theory. Computer scientists have introduced *hybrid automata* [8], a formal model that combines discrete control graphs, usually called *finite state automata*, with continuously evolving variables.

When the evolution of continuous variables are subject to rectangular flow constraints, that is, constraints of the form $\dot{x} \in [a, b]$, hybrid automata are called *rectangular*. For that subclass of hybrid automata, there exists a reasonably efficient algorithm to compute the flow successor [1]. Based on this algorithm, there exists an iterative method that computes the exact set of reachable states when it terminates. This semi-algorithm can be used to establish or refute *safety properties*. On the other hand, if the evolution of continuous variables are subject to more complicated flow constraints, for example affine dynamics like $\dot{x} = 3x - y$, computing the flow successor is more difficult and only approximate methods are known.

⋆ Research fellow supported by the Belgian National Science Foundation (FNRS).

⋆⋆ Supported in part by the AFOSR MURI grant F49620-00-1-0327 and the NSF grants CCR-0208875 and CCR-0225610.

Rectangular hybrid automata have the additional property that they can over-approximate, at any level of precision, the set of behaviors of more complex hybrid automata. A general methodology to analyze complex hybrid automata using rectangular approximations has been proposed in [10]. This methodology can be summarized as follows: to establish a safety property on a complex hybrid automata A, construct a rectangular approximation B of A and check the safety property on B. If the property is established on B, then it also holds on A. If the property is not established on B, then use a more precise rectangular approximation B'.

So far, the construction of the abstraction B and its refinement B' was supposed to be obtained manually. In this paper, we show how to efficiently automate this methodology for the class of affine hybrid automata, that is, hybrid automata whose continuous dynamics are defined by systems of linear differential equations. More precisely, we show (*i*) how to compute automatically rectangular approximations for affine hybrid automata, (*ii*) how to refine automatically and in an optimal way rectangular approximations that fail to establish a given safety property, (*iii*) how to target refinement only to relevant parts of the state space.

Refinements are obtained by splitting location invariants. A split is optimal if it minimizes some measure of the imprecision of the resulting rectangular approximation. Intuitively, the imprecision corresponds to the size of the interval defining the rectangular flow, because the smaller the imprecision, the closer the rectangular dynamics is to the exact dynamics. In this paper, we minimize the maximal imprecision of the dynamics in the split location of the refined automaton.

We have implemented our methodology and applied it to the navigation benchmark proposed in [5], for which we have implemented a prototype. The first results that we have obtained are encouraging. We therefore expect further improvements to the theory of refinement, as well as to the design and implementation of algorithms, to be fruitful, and we plan to continue investigating this field.

Structure of the paper In Section 2 we recall the classical definitions of hybrid automata, giving their semantics as timed transition systems. We show how rectangular overapproximations of affine hybrid automata can be constructed. The methodology to obtain successive refinements of such rectangular approximations is detailed in Section 3 in the form of an optimization problem whose solution is intended to give an optimal way of refining. We give an algorithm to solve the problem in 2 dimensions (when the hybrid systems has 2 variables), and we give an approximation technique for higher dimensions in the form of a more abstract optimization problem that can be solved in general. Finally, the refinement-based verification algorithm is explained in Section 4 and a case study is presented in Section 5. Complete proofs of the theorems can be found at the following web page: `http://www.ulb.ac.be/di/ssd/cfv/publications.html`.

2 Hybrid Automata and Abstraction

Intervals An *interval* is a nonempty convex subset of the set $\mathbb{R}$ of real numbers. For a bounded interval I, we denote the left (resp. right) end-point of I by l_I

(resp. r_I). For an interval I, we define $\mathsf{size}(I) = r_I - l_I$ if I is bounded, and $\mathsf{size}(I) = +\infty$ otherwise. A bounded interval I is *closed* if $l_I \in I$ and $r_I \in I$. Given two closed intervals I_1 and I_2, their *convex hull* is the closed interval $I_3 = I_1 \sqcup I_2$ such that $l_{I_3} = \min(l_{I_1}, l_{I_2})$ and $r_{I_3} = \max(r_{I_1}, r_{I_2})$.

Predicates Let $X = \{x_1, \ldots, x_n\}$ be a finite set of variables. A *valuation* over X is a function $v : X \to \mathbb{R}$. Alternatively, it can be seen as a tuple $(v_1, \ldots, v_n) = (v(x_1), \ldots, v(x_n))$ in $\mathbb{R}^n$. Given $v : X \to \mathbb{R}$ and $Y \subseteq X$, define $v_{|Y} : Y \to \mathbb{R}$ by $v_{|Y}(x) = v(x)\ \forall x \in Y$. A *linear term* over X is an expression of the form $y \equiv a_0 + \sum_{x_i \in X} a_i x_i$ where $a_i \in \mathbb{Q}$ $(0 \leq i \leq n)$ are rational constants. Given a valuation v over X, we write $[\![y]\!]_v$ for the real number $a_0 + \sum_{x_i \in X} a_i v(x_i)$. We denote by $\mathsf{LTerm}(X)$ the set of all linear terms over X. A *rectangular predicate* over X is a formula of the form $\bigwedge_{x \in X} x \in I_x$ where I_x $(x \in X)$ are closed intervals. We denote by $\mathsf{Rect}(X)$ the set of all rectangular predicates over X. Given two rectangular predicates $p = \bigwedge_{x \in X} x \in I_x$ and $q = \bigwedge_{x \in X} x \in J_x$, we define the *size* of p by $|p| = \max_{x \in X}\{\mathsf{size}(I_x)\}$ and the *rectangular hull* of p and q by $p \sqcup q = \bigwedge_{x \in X} x \in I_x \sqcup J_x$. A *linear constraint* over X is a formula of the form $y \bowtie 0$ where $y \in \mathsf{LTerm}(X)$ is a linear term over X and $\bowtie \in \{<, \leq, =, >, \geq\}$. A *linear predicate* over X is a finite conjunction of linear constraints over X. We denote by $\mathsf{Lin}(X)$ the set of all linear constraints over X. For a (rectangular or linear) predicate p over X, we write $[\![p]\!]$ for the set of all valuations $v \in [X \to \mathbb{R}]$ satisfying p. A *polytope* is a closed bounded set $[\![p]\!]$ defined by a linear predicate p. An *affine dynamics predicate* over X is a formula of the form $\bigwedge_{x \in X} \dot{x} = t_x$ where $t_x \in \mathsf{LTerm}(X)$ $(x \in X)$ are linear terms over X and $\dot{x}$ represents the first derivative of x. Let $\dot{X} = \{\dot{x} \mid x \in X\}$. We denote by $\mathsf{Affine}(X, \dot{X})$ the set of all affine dynamics predicates over X. For an affine dynamics predicate p, we write $[\![p]\!]$ for the set of all valuations $v \in [X \cup \dot{X} \to \mathbb{R}]$ satisfying p.

Lines and hyperplanes A function $f : \mathbb{R}^n \to \mathbb{R}$ is called *affine* if it is of the form $f(x) = a_0 + \sum_i a_i x_i$ with $a_i \in \mathbb{Q}$ $(0 \leq i \leq n)$. We say that two affine functions f_1 and f_2 are *parallel* if for some $\lambda \in \mathbb{R}$ the function $f(x) = f_1(x) + \lambda f_2(x)$ is independent of x (that is, ∇f is identically 0). Given a function $f : A \to B$ and $b \in B$, define the *level set* of f corresponding to b by $f^{-1}(b) = \{a \in A \mid f(a) = b\}$. For $C \subseteq B$, define $f^{-1}(C) = \{a \in A \mid f(a) \in C\}$. A *hyperplane* is a level set of an affine function. Given an affine function $f : \mathbb{R}^n \to \mathbb{R}$, we write $\pi \equiv f(x) = 0$ to denote the hyperplane $\pi = f^{-1}(0)$. In $\mathbb{R}^2$ a hyperplane is a *line*. Given two points $a, b \in \mathbb{R}^2$, let $\ell ine(a, b)$ denote the line passing by a and b. We write $[a, b]$ for the *line segment* connecting a and b, *i.e.* the set of convex combinations $\{\lambda a + (1 - \lambda)b \mid 0 \leq \lambda \leq 1\}$.

Transition systems A TTS (*timed transition system*) $\mathcal{T}$ is a tuple $\langle S, S_0, S_f, \Sigma, \rightarrow\rangle$ where S is a (possibly infinite) set of states, $S_0 \subseteq S$ is the set of initial state, $S_f \subseteq S$ is the set of final states, Σ is a finite set of labels including the *silent* label τ, and $\rightarrow \subseteq S \times (\Sigma \cup \mathbb{R}^{\geq 0}) \times S$ is the transition relation. If $(q, \sigma, q') \in \rightarrow$ we write $q \xrightarrow{\sigma} q'$. Define the *successors* of a set $A \subseteq S$ in $\mathcal{T}$ to be the set $\mathsf{Post}_{\mathcal{T}}(A) = \{s' \in S \mid \exists\, s \in A, \exists\, \sigma \in \Sigma \cup \mathbb{R}^{\geq 0} : s \xrightarrow{\sigma} s'\}$. Similarly, the *predecessors* of A in $\mathcal{T}$ are given by $\mathsf{Pre}_{\mathcal{T}}(A) = \{s \in S \mid \exists\, s' \in A, \exists\, \sigma \in \Sigma \cup \mathbb{R}^{\geq 0} : s \xrightarrow{\sigma} s'\}$.

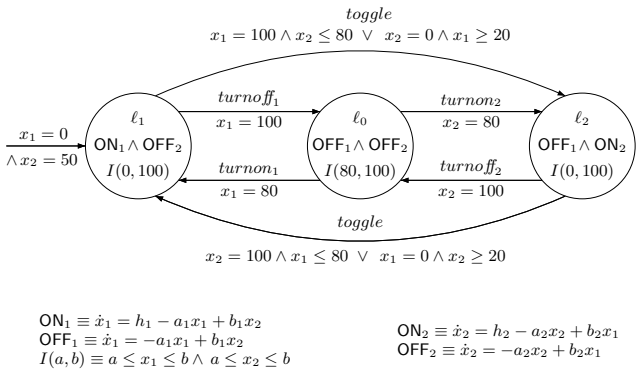


Fig. 1. A shared gas-burner.

The *reachable states* of $\mathcal{T}$ is the set $\mathsf{Reach}(\mathcal{T}) = \bigcup_{i\in\mathbb{N}} \mathsf{Post}^i_{\mathcal{T}}(S_0)$ where $\mathsf{Post}^0_{\mathcal{T}}(A) = A$ and recursively $\mathsf{Post}^i_{\mathcal{T}}(A) = \mathsf{Post}_{\mathcal{T}}(\mathsf{Post}^{i-1}_{\mathcal{T}}(A))$. Let $\mathsf{Reach}^{-1}(\mathcal{T}) = \bigcup_{i\in\mathbb{N}} \mathsf{Pre}^i_{\mathcal{T}}(S_f)$ where $\mathsf{Pre}^0_{\mathcal{T}}(A) = A$ and recursively $\mathsf{Pre}^i_{\mathcal{T}}(A) = \mathsf{Pre}_{\mathcal{T}}(\mathsf{Pre}^{i-1}_{\mathcal{T}}(A))$. Finally, let $\mathsf{Unsafe}(\mathcal{T}) = \mathsf{Reach}(\mathcal{T}) \cap \mathsf{Reach}^{-1}(\mathcal{T})$ be the set of reachable states that can reach the final states. The complement of $\mathsf{Unsafe}(\mathcal{T})$ is the set of safe states.

Given the TTS $\mathcal{T} = \langle S, S_0, S_f, \Sigma, \rightarrow\rangle$, we define the *stutter-closed relation* $\twoheadrightarrow \subseteq S \times (\Sigma\backslash\{\tau\} \cup \mathbb{R}^{\geq 0}) \times S$ as follows: if $\sigma \in \Sigma\backslash\{\tau\}$, then $s \xrightarrow{\sigma}\!\!\!\!\rightarrow s'$ iff there exists a finite sequence $s_0, \ldots, s_k \in S$ of states such that $s = s_0$ and $s_0 \xrightarrow{\tau} s_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} s_k \xrightarrow{\sigma} s'$; if $t \in \mathbb{R}^{\geq 0}$, then $s \xrightarrow{t}\!\!\!\!\rightarrow s'$ iff there exists a finite sequence $s_0, \ldots, s_{2k} \in S$ of states and a finite sequence $t_0, \ldots, t_k \in \mathbb{R}^{\geq 0}$ of constants such that $s = s_0$ and $s_0 \xrightarrow{t_0} s_1 \xrightarrow{\tau} s_2 \xrightarrow{t_1} \ldots \xrightarrow{\tau} s_{2k} \xrightarrow{t_k} s'$ and $t = t_0 + \cdots + t_k$.

Hybrid automata We define two types of hybrid automata, with either affine or rectangular dynamics [1, 11].

Definition 1 [Hybrid automaton] An *hybrid automaton* H is a tuple $\langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final}\rangle$ where: (i) $\mathsf{Loc} = \{\ell_1, \ldots, \ell_m\}$ is a finite set of *locations*. (ii) Lab is a finite set of *labels* containing the silent label τ. (iii) $\mathsf{Edg} \subseteq \mathsf{Loc}\times\mathsf{Lab}\times\mathsf{Loc}$ is a finite set of *edges*. (iv) $X = \{x_1, \ldots, x_n\}$ is a finite set of *variables*. (v) $\mathsf{Init} : \mathsf{Loc} \to \mathsf{Lin}(X)$ gives the *initial condition* $\mathsf{Init}(\ell)$ of location ℓ. The automaton can start in ℓ with an initial valuation v lying in $[\![\mathsf{Init}(\ell)]\!]$. (vi) $\mathsf{Inv} : \mathsf{Loc} \to \mathsf{Lin}(X)$ gives the *invariant condition* $\mathsf{Inv}(\ell)$ of location ℓ. We require that $[\![\mathsf{Inv}(\ell)]\!]$ is bounded for every $\ell \in \mathsf{Loc}$. The automaton can stay in ℓ as long as the values of its variables lie in $[\![\mathsf{Inv}(\ell)]\!]$ (vii) Flow governs the evolution of the variables in each location:

- either $\mathsf{Flow} : \mathsf{Loc} \to \mathsf{Affine}(X, \dot{X})$ and H is called an *affine dynamics* hybrid automaton,
- or $\mathsf{Flow} : \mathsf{Loc} \to \mathsf{Rect}(\dot{X})$ and H is called a *rectangular dynamics* hybrid automaton;

(*viii*) $\mathsf{Jump} : \mathsf{Edg} \to \mathsf{Lin}(X \cup X')$ with $X' = \{x'_1, \ldots, x'_n\}$ gives the *jump condition* $\mathsf{Jump}(e)$ of edge e. The variables in X' refer to the updated values of the variables after the edge has been traversed. (*ix*) $\mathsf{Final} : \mathsf{Loc} \to \mathsf{Lin}(X)$ gives the *final condition* $\mathsf{Final}(\ell)$ of location ℓ. In general, final conditions specify the unsafe states of the system.

Example Fig. 1 represents an affine dynamics hybrid automaton modeling a single gas-burner that is, shared for heating alternatively two water tanks. It has three locations ℓ_0, ℓ_1, ℓ_2 and two variables x_1 and x_2, the temperature in the two tanks. The gas-burner can be either switched off (in ℓ_0) or turned on heating one of the two tanks (in ℓ_1 or ℓ_2). The dynamics in each location is given by a combination of the predicates ON_i and OFF_i ($i = 1, 2$) where the constants a_i model the heat exchange rate of the tank i with the room in which the tanks are located, b_i model the heat exchange rate between the two tanks and h_i depends on the power of the gas-burner. On every edge of the automaton, we have omitted the condition $x'_1 = x_1 \wedge x'_2 = x_2$ also written as $\mathit{stable}(x_1, x_2)$ that asserts that the values of the variables have to be maintained when the edge is traversed. In the sequel, we fix the constants $h_1 = h_2 = 2$, $a_1 = a_2 = 0.01$ and $b_1 = b_2 = 0.005$.

Definition 2 [Semantics of hybrid automata] The *semantics* of an hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$ is the TTS $[\![H]\!] = \langle S, S_0, S_f, \Sigma, \to \rangle$ where $S = \mathsf{Loc} \times \mathbb{R}^n$ is the *state space*, $S_0 = \{(\ell, v) \in S \mid v \in [\![\mathsf{Init}(\ell)]\!]\}$ is the *initial space*, $S_f = \{(\ell, v) \in S \mid v \in [\![\mathsf{Final}(\ell)]\!]\}$ is the *final space*, $\Sigma = \mathsf{Lab}$ and $\to$ contains all the tuples $((\ell, v), \sigma, (\ell', v'))$ such that:

- either there exists $e = (\ell, \sigma, \ell') \in \mathsf{Edg}$ such that $(v, v') \in [\![\mathsf{Jump}(e)]\!]$,
- or $\ell = \ell'$ and $\sigma = \delta \in \mathbb{R}^{\geq 0}$ and there exists a continuously differentiable function $f : [0, \delta] \to [\![\mathsf{Inv}(l)]\!]$ such that $f(0) = v$, $f(\delta) = v'$ and for all $t \in [0, \delta]$:
  - either H has affine dynamics and $(f(t), \dot{f}(t)) \in [\![\mathsf{Flow}(\ell)]\!]$,
  - or H has rectangular dynamics and $\dot{f}(t) \in [\![\mathsf{Flow}(\ell)]\!]$.

  Such a function f is called a *witness* for the transition $((\ell, v), \delta, (\ell', v'))$.

An hybrid automaton H is said *empty* if no final state of H is reachable, that is, if $\mathsf{Reach}([\![H]\!]) \cap \bigcup_{\ell \in \mathsf{Loc}} [\![\mathsf{Final}(\ell)]\!] = \varnothing$. The question to determine if a given hybrid automaton is empty is known as the *emptiness problem*.

To attack this problem, many of the existing tools (*a.o.* HyTech [9], **d/dt** [2], PHAVer [7]) use a symbolic analysis of the hybrid automaton with a forward and/or backward approach: starting from the initial (resp. unsafe) states, iterate the operator $\mathsf{Post}_{[\![H]\!]}$ (resp. $\mathsf{Pre}_{[\![H]\!]}$) until a fix point is reached and then check emptiness of the intersection with the unsafe (resp. initial) states. Those procedures are not guaranteed to terminate since the emptiness problem is undecidable for both affine and rectangular dynamics hybrid automata [11].

Remark The more efficient tool PHAVer does not implement the backward analysis. Nevertheless, for a rectangular dynamics automaton H it is possible to define the *reverse automaton* $-H$ such that $\mathsf{Pre}_{[\![H]\!]} = \mathsf{Post}_{[\![-H]\!]}$, and so such that $\mathsf{Reach}^{-1}_{[\![H]\!]} = \mathsf{Reach}_{[\![-H]\!]}$. Roughly, the construction consists in reversing

the flow dynamics (an interval $[a, b]$ is replaced by the interval $[-b, -a]$) and the jump conditions (by permuting primed and unprimed variables). The other components are kept unchanged except the initial and unsafe sets which are swapped [11].

Overapproximations The formal link between an automaton and its approximation will be drawn with the notion of *weak simulation*[12].

Definition 3 [Weak simulation] Given two TTS $\mathcal{T}_1 = \langle S^1, S_0^1, S_f^1, \Sigma, \rightarrow^1 \rangle$ and $\mathcal{T}_2 = \langle S^2, S_0^2, S_f^2, \Sigma, \rightarrow^2 \rangle$ with the same set of labels, we write $\mathcal{T}_1 \succeq \mathcal{T}_2$ and say that $\mathcal{T}_1$ *weakly simulates* $\mathcal{T}_2$ if there exists a relation $R \subseteq S^1 \times S^2$ such that:

1. for all $(s_1, s_2) \in R$, for each $\sigma \in \Sigma \backslash \{\tau\} \cup \mathbb{R}^{\geq 0}$, if $s_2 \stackrel{\sigma}{\rightarrow\!\!\!\!\rightarrow} s_2'$, then there exists $s_1' \in S^1$ such that $s_1 \stackrel{\sigma}{\rightarrow\!\!\!\!\rightarrow} s_1'$ and $(s_1', s_2') \in R$,
2. for all $s_2 \in S_0^2$, there exists $s_1 \in S_0^1$ such that $(s_1, s_2) \in R$,
3. and for all $s_2 \in S_f^2$, for all $s_1 \in S^1$, if $(s_1, s_2) \in R$, then $s_1 \in S_f^1$.

Such a relation R is called a *simulation relation* for $\mathcal{T}_1 \succeq \mathcal{T}_2$.

For a relation $S \subseteq S^1 \times S^2$, let the inverse of S be the relation $S^{-1} = \{(s_2, s_1) \mid (s_1, s_2) \in S\}$.

Definition 4 [Weak bisimulation] Two TTS $\mathcal{T}_1$ and $\mathcal{T}_2$ are *weakly bisimilar* (noted $\mathcal{T}_1 \approx \mathcal{T}_2$) if there exists a simulation relation R for $\mathcal{T}_1 \succeq \mathcal{T}_2$ and a simulation relation S for $\mathcal{T}_2 \succeq \mathcal{T}_1$ such that $R = S^{-1}$.

In the sequel, we also use a slightly different notion of simulation where only the unsafe states are to be simulated, because of the following observation: a state that has been proven to be safe in $\mathcal{T}_1$ is necessarily safe in $\mathcal{T}_2$ if $\mathcal{T}_1 \succeq \mathcal{T}_2$ and therefore the emptiness problem for an hybrid automaton H has the same answer as the emptiness problem on H with some of its safe states forbidden. So, it is sufficient to refine the automaton in its unsafe states. More details are given in Section 4.

Definition 5 [Weak simulation for unsafe behaviours] Given two TTS $\mathcal{T}_1 = \langle S^1, S_0^1, S_f^1, \Sigma, \rightarrow^1 \rangle$ and $\mathcal{T}_2 = \langle S^2, S_0^2, S_f^2, \Sigma, \rightarrow^2 \rangle$ with the same set of labels, we write $\mathcal{T}_1 \succeq_{unsafe} \mathcal{T}_2$ and say that $\mathcal{T}_1$ *weakly simulates the unsafe behaviours of* $\mathcal{T}_2$ if there exists a relation $R \subseteq S^1 \times \mathsf{Unsafe}(\mathcal{T}_2)$ such that:

1. for all $(s_1, s_2) \in R$, for each $\sigma \in \Sigma \backslash \{\tau\} \cup \mathbb{R}^{\geq 0}$, if $s_2 \stackrel{\sigma}{\rightarrow\!\!\!\!\rightarrow} s_2'$ and $s_2' \in \mathsf{Unsafe}(\mathcal{T}_2)$, then there exists $s_1' \in S^1$ such that $s_1 \stackrel{\sigma}{\rightarrow\!\!\!\!\rightarrow} s_1'$ and $(s_1', s_2') \in R$,
2. for all $s_2 \in S_0^2$, there exists $s_1 \in S_0^1$ such that $(s_1, s_2) \in R$,
3. and for all $s_2 \in S_f^2$, for all $s_1 \in S^1$, if $(s_1, s_2) \in R$, then $s_1 \in S_f^1$.

Such a relation R is called a *simulation relation* for $\mathcal{T}_1 \succeq_{unsafe} \mathcal{T}_2$.

Definition 6 [Weak bisimulation for unsafe behaviours] Two TTS $\mathcal{T}_1$ and $\mathcal{T}_2$ are *weakly bisimilar for unsafe behaviours* (noted $\mathcal{T}_1 \approx_{unsafe} \mathcal{T}_2$) if there exists a simulation relation R for $\mathcal{T}_1 \succeq_{unsafe} \mathcal{T}_2$ and a simulation relation S for $\mathcal{T}_2 \succeq_{unsafe} \mathcal{T}_1$ such that $R = S^{-1}$.

Lemma 7 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two* TTS*. If $\mathcal{T}_1 \succeq \mathcal{T}_2$, then $\mathcal{T}_1 \succeq_{unsafe} \mathcal{T}_2$.*

Theorem 8 *Let H and H' be two hybrid automata such that $[\![H']\!] \succeq_{unsafe} [\![H]\!]$. If H' is empty, then so is H.*

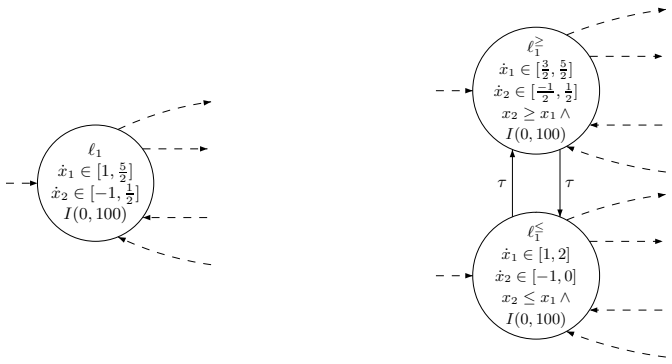
Definition 9 [Rectangular phase-portrait approximation] We say that a hybrid automaton H' is a *rectangular phase-portrait approximation* of an hybrid automaton H if $[\![H']\!] \succeq [\![H]\!]$ and H' has rectangular dynamics.

A natural way for constructing a rectangular phase-portrait approximation $H' = \mathsf{rect}(H)$ of an affine dynamics hybrid automaton H is to replace in each location ℓ of H the affine flow condition $\mathsf{Flow}_H(\ell) = \bigwedge_{x \in X} \dot{x} = t_x$ by the rectangular predicate $\mathsf{Flow}_{H'}(\ell) = \bigwedge_{x \in X} \dot{x} \in I_x$ where I_x is the tightest interval containing the set $\{[\![t_x]\!]_v \mid v \in [\![\mathsf{Inv}(l)]\!]\}$. The bounds of I_x can be determined by a linear program since t_x is a linear term and $\mathsf{Inv}(l)$ is a linear predicate. The proof that $[\![H']\!] \succeq [\![H]\!]$ is obvious since for any $v \in [\![\mathsf{Flow}_H(\ell)]\!]$ we have $v_{|\dot{X}} \in [\![\mathsf{Flow}_{H'}(\ell)]\!]$.

Example Fig. 2(a) depicts the result of this construction when applied to the location ℓ_1 of the shared gas-burner automaton.

For an hybrid automaton H with set of locations Loc_H, let $\mathsf{SafeLoc}(H) = \{\ell \in \mathsf{Loc}_H \mid \nexists (\ell, v) \in \mathsf{Unsafe}([\![H]\!])\}$.

Lemma 10 *For every affine dynamics hybrid automaton H, we have* $\mathsf{SafeLoc}(\mathsf{rect}(H)) \subseteq \mathsf{SafeLoc}(H)$.



(a) Rectangular phase-portrait.

(b) Location splitting.

Fig. 2. Location ℓ_1 of the shared gas-burner.

3 Abstraction Refinement for Hybrid Automata

Definition 11 [Refined approximation] Given H' and H'' two rectangular phase-portrait approximations of an hybrid automaton H, we say that H'' *refines* H' if $[\![H']\!] \succeq [\![H'']\!]$.

A natural way of refining an approximation of an affine dynamics hybrid automaton is to *split* its locations by partitioning or covering their invariant.

Definition 12 [Cut] Given a polytope $P \subseteq \mathbb{R}^n$ and a hyperplane $\pi \equiv f(x) = 0$, we define the *cut* $P/\pi = \langle P^+, P^- \rangle$ where $P^+ = P \cap f^{-1}(\mathbb{R}^{\geq 0})$ and $P^- = P \cap f^{-1}(\mathbb{R}^{\leq 0})$. The cut P/π is said *non-trivial* if $P^+ \neq \varnothing$ and $P^- \neq \varnothing$.

Thus a non-trivial cut P/π of a polytope P is a *cover* of P but not a partition since the two pieces are closed sets and they share the points in $P \cap \pi$.

Definition 13 [Location splitting] Given an hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$, one of its locations $\ell^\star \in \mathsf{Loc}$ and a hyperplane $\pi \equiv f_\pi(x) = 0$ in $\mathbb{R}^{|X|}$, the *splitting* of H by the hyperplane π in location $\ell^\star$ is the hybrid automaton $\mathsf{split}(H, \ell^\star, \pi) = \langle \mathsf{Loc}', \mathsf{Lab}', \mathsf{Edg}', X', \mathsf{Init}', \mathsf{Inv}', \mathsf{Flow}', \mathsf{Jump}', \mathsf{Final}' \rangle$ where: (i) $\mathsf{Loc}' = \mathsf{Loc} \backslash \{\ell^\star\} \cup \{(\ell^\star, P), (\ell^\star, Q)\}$ where $P = \mathsf{Inv}(\ell^\star) \wedge f_\pi(x) \leq 0$ and $Q = \mathsf{Inv}(\ell^\star) \wedge f_\pi(x) \geq 0$. For $\ell' \in \mathsf{Loc}'$, let $loc(\ell') = \ell'$ if $\ell' \in \mathsf{Loc}$ and $loc(\ell') = \ell^\star$ otherwise. (ii) $\mathsf{Lab}' = \mathsf{Lab}$. (iii) $\mathsf{Edg}' = E_1 \cup E_2$ where $E_1 = \{(\ell, \sigma, \ell') \mid \ell, \ell' \in \mathsf{Loc}' \wedge (loc(\ell), \sigma, loc(\ell')) \in \mathsf{Edg}\}$ is the set of edges inherited from H and $E_2 = \{(\ell, \tau, \ell') \mid \ell, \ell' \in \mathsf{Loc}' \wedge loc(\ell) = loc(\ell') = \ell^\star\}$ are silent edges between the two copies of the location $\ell^\star$. (iv) $X' = X$. (v) $\mathsf{Init}'(\ell') = \mathsf{Init}(loc(\ell'))$ for each $\ell' \in \mathsf{Loc}'$. (vi) $\mathsf{Inv}'(\ell') = \mathsf{Inv}(\ell')$ for each $\ell' \in \mathsf{Loc} \backslash \{\ell^\star\}$, $\mathsf{Inv}'(\ell^\star, P) = P$ and $\mathsf{Inv}'(\ell^\star, Q) = Q$. (vii) $\mathsf{Flow}'(\ell') = \mathsf{Flow}(loc(\ell'))$ for each $\ell' \in \mathsf{Loc}'$. $(viii)$ for every $e = (\ell, \sigma, \ell') \in E_1$ we have $\mathsf{Jump}'(e) = \mathsf{Jump}(loc(\ell), \sigma, loc(\ell))$, and for every $e \in E_2$ we have $\mathsf{Jump}'(e) = \mathit{stable}(X)$. (ix) $\mathsf{Final}'(\ell') = \mathsf{Final}(loc(\ell'))$ for each $\ell' \in \mathsf{Loc}'$.

Example Fig. 2(b) shows the rectangular phase-portrait approximation of the splitting of the location ℓ_1 of the shared gas-burner by the line $x_1 = x_2$. The resulting automaton is a refinement since the ranges of the rectangular dynamics have decreased in each of the two splitted locations.

This technique is very general and has been applied to hybrid automata with nonlinear dynamics [10]. However, in that last reference, the proof of correctness (that the refined automaton $\mathsf{split}(H, \ell^\star, \pi)$ weakly simulates the original automaton H) relies crucially on the fact that the split of an invariant is derived from a finite *open cover*, that is, a location $\ell^\star$ is replaced by $(\ell^\star, P)$ and $(\ell^\star, Q)$ where P, Q are open sets such that $\mathsf{Inv}(\ell^\star) \subseteq P \cup Q$. Unfortunately, the proof cannot be extended to closed covers: for example, the continuously differentiable function $f : \mathbb{R}^{\geq 0} \to \mathbb{R}$ defined by $f(0) = 0$ and $f(t) = t^2 \sin(1/t)$ for $t > 0$, oscillates infinitely in every interval $[0, \epsilon]$ ($\epsilon > 0$). So that if $f_{|[0,\delta]}$ was the witness of a transition $((\ell, v), \delta, (\ell, v'))$ of an automaton H with variable $X_H = \{y\}$, it would be impossible for the automaton $\mathsf{split}(H, \ell, y = 0)$ to mimic that transition since time cannot progress by any amount while maintaining either $y \geq 0$ or $y \leq 0$. However, this kind of pathological behaviour can not appear as a solution of

a system of affine dynamics flow conditions (even though spiral trajectories are still possible) because such solutions are *analytic* and zeroes of analytic functions are isolated.

Theorem 14 *For every hybrid automaton H, for every of its location ℓ, and every hyperplane π, we have $[\![\mathsf{split}(H,\ell,\pi)]\!] \approx [\![H]\!]$, that is, $[\![\mathsf{split}(H,\ell,\pi)]\!]$ and $[\![H]\!]$ are weakly bisimilar.*

With the remark above, the proof of Theorem 14 is straightforward.

In this section, we are interested in finding *automatically* an optimal cut for refining the state space. We consider the general problem to split a location in an optimal way, that is, to minimize the imprecision of the resulting rectangular phase-portrait approximation. The definition of the imprecision could have several forms. We decide to minimize the maximal size of the rectangular predicates that appear as flow conditions in the rectangular approximation of the splitted automaton (and particularly in the splitted location). It may be necessary to scale the variables in order to give sense to the comparison of their dynamics range. We now discuss our choice. On the one hand, this criterion is natural since the precision of the approximation is directly connected to the size of the rectangular predicates. Further, minimizing the maximal size ensures that the approximation becomes more precise. On the other hand, other criteria we have tried (minimizing the sum of the squares of the sizes, minimizing the size of the reachable set, etc.) gave rise to computational difficulties due to their non-linear form. Our criterion can be handled with linear programming techniques, and showed practical applicability with promising results (see Section 5).

3.1 Optimization criteria

We ask to split the invariant of a location into two convex pieces, minimizing the maximal size of the flow intervals obtained in the approximation. We define two versions of this problem, one called *concrete* when the given dynamics is affine on the invariant, and one called *abstract* when the invariant is already covered by a number of pieces each with its rectangular dynamics. The second formulation is introduced because we have no simple algorithm in $n\mathsf{D}$ for $n \geq 3$ for solving the concrete problem, while we have a general algorithm for the abstract one. Therefore, it may be of interest to discretize the original affine system in order to get an approximation of the best split. We show that the result can be made arbitrarily close to the exact solution.

In Definition 15, we associate to each subset $Q \subseteq P$ of the invariant P of a location the tightest rectangular dynamics that contains the exact dynamics in the set Q (which is defined by an affine predicate of the form $\bigwedge_{x_i \in X} \dot{x_i} = f_i(x_1, \ldots, x_n)$). Then, the *imprecision* of a cut of P into two pieces is defined to be the maximal size of the rectangular predicates associated to each piece.

Definition 15 [Concrete imprecision] Let $X = \{x_1, \ldots, x_n\}$ be a finite set of variables. Let $P \subset \mathbb{R}^n$ be a polytope and $F = \{f_1, \ldots, f_n\}$ a set of n affine functions $f_i : \mathbb{R}^n \to \mathbb{R}$ $(1 \leq i \leq n)$. For a polytope $Q \subseteq P$, we define the rectangular predicate $\mathsf{range}^F(Q) = \bigwedge_{x \in X} x \in I_x$ where for each $x_i \in X$, we have $I_{x_i} = f_i(Q)$. We define the *concrete imprecision* of a cut $P/\pi = \langle P^+, P^- \rangle$ by $\mathsf{sizeRange}^F(P/\pi) = \max\left\{|\mathsf{range}^F(P^+)|, |\mathsf{range}^F(P^-)|\right\}$.

Definition 16 [Concrete optimal-cut problem] An *instance of the concrete optimal-cut problem* is a tuple $\langle P, X, F\rangle$ where: $P \subset \mathbb{R}^n$ is a polytope, $X = \{x_1, \ldots, x_n\}$ is a set of n variables, and F is a set of n affine functions $f_i : \mathbb{R}^n \to \mathbb{R}$ $(1 \leq i \leq n)$.

The *concrete optimal-cut problem* is to determine a hyperplane $\pi^\star \subset \mathbb{R}^n$ such that for every hyperplane $\pi \subset \mathbb{R}^n$, $\mathsf{sizeRange}^F(P/\pi^\star) \leq \mathsf{sizeRange}^F(P/\pi)$.

We define an abstract version of the optimal-cut problem where the invariants of the locations are originally given as a union of polytopes $\mathcal{P}$, and the flow condition in each piece is rectangular. This form of the problem is obtained when we discretize the concrete problem according to $\mathcal{P}$. The abstract problem asks to split the location into only *two* pieces such that the maximal flow interval in the two pieces is minimized.

Definition 17 [Abstract imprecision] Let $X = \{x_1, \ldots, x_n\}$ be a finite set of variables. Let $P \subset \mathbb{R}^n$ be a polytope covered by a finite set of polytopes $\mathcal{P} = \{P_1, \ldots, P_m\}$, that is, such that $P = P_1 \cup \cdots \cup P_m$. Let $\mathsf{Flow} : \mathcal{P} \to \mathsf{Rect}(\dot{X})$ be a function that associates to each piece of $\mathcal{P}$ a rectangular dynamics. For a polytope $Q \subseteq P$, we define the rectangular predicate $\mathsf{range}^{\mathsf{Flow}}(Q) = \bigsqcup_{P_j \in \mathcal{P}, P_j \cap Q \neq \varnothing} \mathsf{Flow}(P_j)$. We define the *abstract imprecision* of a cut $P/\pi = \langle P^+, P^-\rangle$ by $\mathsf{sizeRange}^{\mathsf{Flow}}(P/\pi) = \max\left\{|\mathsf{range}^{\mathsf{Flow}}(P^+)|, |\mathsf{range}^{\mathsf{Flow}}(P^-)|\right\}$.

Definition 18 [Abstract optimal-cut problem] An *instance of the abstract optimal-cut problem* is a tuple $\langle P, \mathcal{P}, X, \mathsf{Flow}\rangle$ where: $P \subset \mathbb{R}^n$ is a polytope, $\mathcal{P} = \{P_1, \ldots, P_m\}$ is finite set of polytopes such that $P = P_1 \cup \cdots \cup P_m$, $X = \{x_1, \ldots, x_n\}$ is a set of n variables, and $\mathsf{Flow} : \mathcal{P} \to \mathsf{Rect}(\dot{X})$.

The *abstract optimal-cut problem* asks to determine a hyperplane $\pi^\star \subset \mathbb{R}^n$ such that for every hyperplane $\pi \subset \mathbb{R}^n$, $\mathsf{sizeRange}^{\mathsf{Flow}}(P/\pi^\star) \leq \mathsf{sizeRange}^{\mathsf{Flow}}(P/\pi)$.

3.2 Solution of the abstract optimal-cut problem

We give an algorithm for solving the abstract problem and show how it can be used to approximate the solution of the concrete problem.

Definition 19 [Separability] Two sets $A, B \subseteq \mathbb{R}^n$ are *separable* if there exists an affine function $f : \mathbb{R}^n \to \mathbb{R}$ such that $\forall x \in A : f(x) \leq 0$ and $\forall x \in B : f(x) \geq 0$.

This definition extends to sets of sets: we say that $\mathcal{A} \subseteq 2^{\mathbb{R}^n}$ and $\mathcal{B} \subseteq 2^{\mathbb{R}^n}$ are separable if $\bigcup \mathcal{A}$ and $\bigcup \mathcal{B}$ are separable. Separability can be tested using the *convex hull* of a set, denoted $\mathcal{C}Hull(\cdot)$.

Lemma 20 *Two sets $A, B \subseteq \mathbb{R}^n$ are separable iff there exists a hyperplane $\pi \subset \mathbb{R}^n$ such that $\mathcal{C}Hull(A) \cap \mathcal{C}Hull(B) \subseteq \pi$.*

An optimal cut for the abstract problem $\langle P, \mathcal{P}, X, \mathsf{Flow}\rangle$ is solved by Algorithm 1. It uses two external functions *value* and *separable* taking as input two sets $\mathcal{A}$ and $\mathcal{B}$ of polytopes. The function *value* returns the number $\mathsf{sizeRange}^{\mathsf{Flow}}(\langle \bigcup \mathcal{A}, \bigcup \mathcal{B}\rangle)$ and the boolean function *separable* returns true iff $\mathcal{A}$ and $\mathcal{B}$ are separable. Lemma 20 suggests a natural implementation of *separable*.

This algorithm constructs two sets of pieces $G^+ \subseteq \mathcal{P}$ and $G^- \subseteq \mathcal{P}$ to be separated from each other, and maintains a set $G_0 = \mathcal{P} \backslash (G^+ \cup G^-)$ of untreated pieces. Initially, we have $G^+ = G^- = \varnothing$ and $G_0 = \mathcal{P}$. The call $split(\varnothing, \varnothing, \mathcal{P})$ returns two sets G^+ and G^- that are separable and such that any separating hyperplane of G^- and G^+ is an optimal cut for $\langle P, \mathcal{P}, X, \mathsf{Flow} \rangle$. Intuitively, the function *split* iteratively selects two pieces $P_i, P_j \in \mathcal{P}$ that are forced to be separated because their flow interval $\mathsf{range}^{\mathsf{Flow}}(P_i \cup P_j)$ is maximal. The separation constraint can be represented as an edge between P_i and P_j in a graph whose vertices is the set $\mathcal{P}$. We can add new such constraints while the graph is 2-colorable with the additional requirement that the two sets of pieces induced by the 2-coloring is physically separable by a hyperplane. In the case the new edge is already connected to the rest of the graph, the color of the common vertex imposes the color of the other. Otherwise, the algorithm has to explore two choices (corresponding to put P_i in G^- and P_j in G^+ or vice versa). An obvious argument shows that this can occur only n times (where $n = |X|$ is the number of variables) so that the the algorithm is in $\mathrm{O}(m.2^n)$, assuming constant execution time of external functions. We do not know if this bound is tight for the problem.

Theorem 21 *Algorithm 1 is correct and always terminates.*

The Algorithm 1 can be used to solve the concrete optimal-cut problem up to a precision $\epsilon \in \mathbb{Q}^{>0}$, as stated by Theorem 23. It suffices to discretize the polytope with a grid of size ϵ: given a set $F = \{f_1, \ldots, f_n\}$ of n affine functions in $\mathbb{R}^n$, let $\mathsf{Grid}_\epsilon^F = \{\bigcap_{1 \leq i \leq n} f_i^{-1}([k_i\,\epsilon, (k_i + 1)\,\epsilon]) \mid (k_1, \ldots, k_n) \in \mathbb{Z}^n\}$. In practice, the complexity blows up since the number of elements in the grid is exponential in $1/\epsilon$.

Definition 22 [ϵ-discretization of a concrete optimal-cut problem] Let $Q = \langle P, X, F \rangle$ be an instance of the concrete optimal-cut problem in $\mathbb{R}^n$, and let $\epsilon \in \mathbb{Q}^{>0}$. The *$\epsilon$-discretization* of Q is the instance $Q_\epsilon = \langle P, \mathcal{P}, X, \mathsf{Flow} \rangle$ of the abstract optimal-cut problem such that:

- $\mathcal{P} = \{P \cap box \mid box \in \mathsf{Grid}_\epsilon^F \wedge P \cap box \neq \varnothing\}$. Notice that $\mathcal{P}$ is finite since P is bounded;
- for each $P_j \in \mathcal{P}$, we have $\mathsf{Flow}(P_j) = \mathsf{range}^F(P_j)$ which is a rectangular predicate of size at most ϵ.

Theorem 23 *Let $Q = \langle P, X, F \rangle$ be an instance of the concrete optimal-cut problem and let $Q_\epsilon = \langle P, \mathcal{P}, X, \mathsf{Flow} \rangle$ be its ϵ-discretization for some $\epsilon \in \mathbb{Q}^{>0}$. If $\pi^\star$ is a solution for Q and $\pi_\epsilon^\star$) is a solution for Q_ϵ, then $\mathsf{sizeRange}^F(P/\pi_\epsilon^\star) - \mathsf{sizeRange}^F(P/\pi^\star) < \epsilon$.*

3.3 Solution of the concrete optimal-cut problem in $\mathbb{R}^2$

We propose an algorithm to solve the concrete optimal-cut problem $\langle P, X, F \rangle$ in two dimensions ($P \subset \mathbb{R}^2$) when $F = \{f_1, f_2\}$ contains two functions. It is shown as Algorithm 2. This algorithm is inspired by the abstract Algorithm 1 applied

Algorithm 1: Algorithm for the abstract optimal-cut problem.

```
   Input  : An instance ⟨P, 𝒫, X, Flow⟩ of the abstract optimal-cut problem.
   Result : Two separable sets G⁻ and G⁺ such that any separating hyperplane of G⁻ and
            G⁺ is an optimal cut for ⟨P, 𝒫, X, Flow⟩.
   begin
       return split(∅, ∅, 𝒫) ;
   end

   external function value(𝒜,ℬ: set of polytopes): ℝ^{≥0}
   external function separable(𝒜,ℬ: set of polytopes): {true, false}

   function split(G⁻, G⁺, G₀: set of polytopes): 2^𝒫 × 2^𝒫 × ℝ^{≥0} begin
1      Let P_i, P_j ∈ 𝒫 maximizing range^Flow(P_i ∪ P_j) subject to P_i ∈ G₀ ∨ P_j ∈ G₀ ;
2      if no such P_i, P_j exists then return ⟨G⁻, G⁺, value(G⁻ ∪ G₀, G⁺ ∪ G₀)⟩ ;
3      if P_i ∈ G₀ ∧ P_j ∈ G₀ then
4          v_A ← ∞ ;
5          v_B ← ∞ ;
6          if separable(G⁻ ∪ {P_i}, G⁺ ∪ {P_j}) then
7              ⟨A₁, A₂, v_A⟩ ← split(G⁻ ∪ {P_i}, G⁺ ∪ {P_j}, G₀\{P_i, P_j}) ;
8          if separable(G⁻ ∪ {P_j}, G⁺ ∪ {P_i}) then
9              ⟨B₁, B₂, v_B⟩ ← split(G⁻ ∪ {P_j}, G⁺ ∪ {P_i}, G₀\{P_i, P_j}) ;
10         if v_A = v_B = ∞ then return ⟨G⁻, G⁺, value(G⁻ ∪ G₀, G⁺ ∪ G₀)⟩ ;
11         if v_A ≤ v_B then
12             return ⟨A₁, A₂, v_A⟩ ;
           else
13             return ⟨B₁, B₂, v_B⟩ ;
       else
14         Assume w.l.o.g. that P_i ∈ G⁻ ;
15         if separable(G⁻, G⁺ ∪ {P_j}) then
16             return split(G⁻, G⁺ ∪ {P_j}, G₀\{P_j}) ;
           else
17             return ⟨G⁻, G⁺, value(G⁻ ∪ G₀, G⁺ ∪ G₀)⟩ ;
   end
```

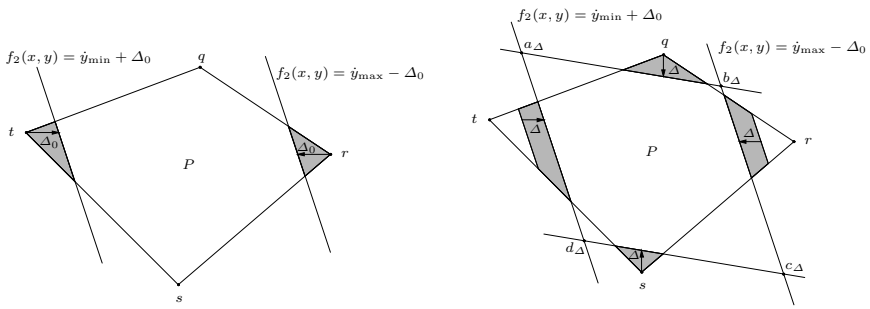
to an ϵ-discretization of the concrete problem with $\epsilon \to 0$. The main trick is to translate the condition of separability expressed with convex hulls into a more continuous condition. We show that this condition can be expressed as a linear program.

Let us execute Algorithm 2 and explain informally why it is correct. The input is an instance $\langle P, X, \{f_1, f_2\}\rangle$ of the concrete optimal-cut problem. We represent P on Fig. 3(a).

At lines 1,2 we compute the interval image of P by f_1 and f_2, and the size of the those intervals (r_x and r_y). The assumption of line 3 implies that the points r and t on Fig. 3(a) are such that $\mathsf{sizeRange}^F(\{r,t\}) = \mathsf{sizeRange}^F(P)$. Therefore, any cut that separates those points is better than any other cut. This remains true for the shaded regions defined by Δ_0 on Fig. 3(a) until:

- either Δ_0 becomes equal to $r_y - r_x$,
- or Δ_0 reaches the value $\frac{r_y}{2}$, and the optimal-cut is given by the line $\ell \equiv f_2(x,y) = \dot{y}_{\min} + \frac{r_y}{2}$.

This alternative is tested at line 4: the condition $r_y \geq 2\,r_x$ is equivalent to $\frac{r_y}{2} \leq r_y - r_x$. If $r_y < 2\,r_x$, the algorithm continues as depicted on Fig. 3(b), separating all pairs of points that give the largest range for function f_1 and f_2.



(a) First phase. (b) Second Phase.

Fig. 3. Algorithm 2.

The four regions P_q, P_r, P_s and P_t (containing respectively q, r, s and t) are growing altogether at the same "rate". The algorithm will stop whenever it becomes impossible to separate both P_q from P_s and P_r from P_t. As in the abstract algorithm, there are two branches to explore, corresponding to separate either $\{P_q, P_r\}$ from $\{P_s, P_t\}$ or $\{P_q, P_t\}$ from $\{P_r, P_s\}$. On Fig. 3(b), is represented the intersection of the level sets of f_1 and f_2 as $a_\Delta, b_\Delta, c_\Delta$ and d_Δ. The subscript emphasizes the fact that those points are moving when Δ varies.

Intuitively, $\{P_q, P_r\}$ and $\{P_s, P_t\}$ are separable iff a_Δ and c_Δ are outside P, a possible separating line being the line connecting a_Δ and c_Δ. Similarly, $\{P_q, P_t\}$ and $\{P_r, P_s\}$ are separable iff b_Δ and d_Δ are outside P. Assume that, as Δ increases, one of the points b_Δ and d_Δ first enters P. Then, it becomes impossible to separate $\{P_q, P_t\}$ from $\{P_r, P_s\}$. But since $\{P_q, P_r\}$ and $\{P_s, P_t\}$ are still separable, the algorithm can continue. When either a_Δ or c_Δ enters P, the algorithm stops (with say $\Delta = \Delta^\star$). An optimal line cut is given by the line passing by $a_{\Delta^\star}$ and $c_{\Delta^\star}$.

As it can be shown, for $z = a \dots d$ the point z_Δ enters P for $\Delta_z = \min\{\Delta \mid z_\Delta \in P\}$. Then $\Delta^\star = \max(\min(\Delta_a, \Delta_c), \min(\Delta_b, \Delta_d))$. Finally, notice that there are several other special configurations that require a particular treatment by Algorithm 2 (lines 13–33).

Remark The macro-instructions of Algorithm 2 (lines 1, 11, 14) can be seen as linear programs. In particular, the coordinates of a_Δ, b_Δ, c_Δ and d_Δ are linear in the parameter Δ so that the minimization of line 11 is a linear program.

Theorem 24 *Algorithm 2 is correct and always terminates.*

Algorithm 2 applies to the optimal-cut problem in 2D. A straightforward extension to higher dimension is not trivially feasible. Indeed, already in 3D, we have examples showing that the separability is not guaranteed even when the level sets (which are planes) are intersecting entirely outside the given polytope.

Algorithm 2: Algorithm for computing the concrete optimal-cut in 2D.

Input : An instance $S = \langle P, X, F\rangle$ of the concrete optimal-cut problem with $P \subset \mathbb{R}^2$, $X = \{x, y\}$, and $F = \{f_1, f_2\}$.

Result : A line that solves the concrete optimal-cut problem for S.

begin

$[\dot{x}_{\min}, \dot{x}_{\max}] \leftarrow f_1(P)$; $[\dot{y}_{\min}, \dot{y}_{\max}] \leftarrow f_2(P)$;

$r_x \leftarrow \dot{x}_{\max} - \dot{x}_{\min}$; $r_y \leftarrow \dot{y}_{\max} - \dot{y}_{\min}$;

Assume w.l.o.g. that $r_y \geq r_x$;

if $r_y \geq 2\, r_x$ **then return** $\ell \equiv f_2(x, y) = \dot{y}_{\min} + \frac{r_y}{2}$;

$\Delta_0 \leftarrow r_y - r_x$;

Let Δ be a symbolic parameter ;

$a_\Delta \leftarrow f_2^{-1}(\dot{y}_{\min} + \Delta_0 + \Delta) \cap f_1^{-1}(\dot{x}_{\min} + \Delta)$;

$b_\Delta \leftarrow f_1^{-1}(\dot{x}_{\min} + \Delta) \cap f_2^{-1}(\dot{y}_{\max} - \Delta_0 - \Delta)$;

$c_\Delta \leftarrow f_2^{-1}(\dot{y}_{\max} - \Delta_0 - \Delta) \cap f_1^{-1}(\dot{x}_{\max} - \Delta)$;

$d_\Delta \leftarrow f_1^{-1}(\dot{x}_{\max} - \Delta) \cap f_2^{-1}(\dot{y}_{\min} + \Delta_0 + \Delta)$;

for $z = a$ ***to*** d **do** $\Delta_z \leftarrow \min\{\Delta \mid z_\Delta \in P\}$;

$\Delta_1 \leftarrow \min(\Delta_a, \Delta_c)$; $\Delta_2 \leftarrow \min(\Delta_b, \Delta_d)$;

if $\Delta_1 \geq \frac{r_y}{2} - \Delta_0 \ \vee \ \Delta_2 \geq \frac{r_y}{2} - \Delta_0$ **then return** $\ell \equiv f_2(x, y) = \dot{y}_{\min} + \frac{r_y}{2}$;

$Q_{\min} \leftarrow P \cap f_1^{-1}(\dot{x}_{\min})$; $Q_{\max} \leftarrow P \cap f_1^{-1}(\dot{x}_{\max})$;

if $f_2(Q_{\min}) \cap [\dot{y}_{\min}, \dot{y}_{\min} + \Delta_0] \neq \varnothing \ \wedge \ f_2(Q_{\min}) \cap [\dot{y}_{\max} - \Delta_0, \dot{y}_{\max}] \neq \varnothing$ **then**

  return $\ell \equiv f_2(x, y) = \dot{y}_{\min} + \frac{r_y}{2}$;

else if $f_2(Q_{\min}) \cap [\dot{y}_{\min}, \dot{y}_{\min} + \Delta_0] \neq \varnothing \neq \varnothing$ **then**

  if $f_2(Q_{\max}) \cap [\dot{y}_{\min}, \dot{y}_{\min} + \Delta_0] \neq \varnothing$ **then**

    return $\ell \equiv f_2(x, y) = \dot{y}_{\min} + \frac{r_y}{2}$;

  else

    return $\ell ine(b_{\Delta_2}, d_{\Delta_2})$;

else if $f_2(Q_{\min}) \cap [\dot{y}_{\max} - \Delta_0, \dot{y}_{\max}] \neq \varnothing$ **then**

  if $f_2(Q_{\max}) \cap [\dot{y}_{\max} - \Delta_0, \dot{y}_{\max}] \neq \varnothing$ **then**

    return $\ell \equiv f_2(x, y) = \dot{y}_{\min} + \frac{r_y}{2}$;

  else

    return $\ell ine(a_{\Delta_1}, c_{\Delta_1})$;

else if $f_2(Q_{\max}) \cap [\dot{y}_{\max} - \Delta_0, \dot{y}_{\max}] \neq \varnothing \ \wedge \ f_2(Q_{\max}) \cap [\dot{y}_{\min}, \dot{y}_{\min} + \Delta_0] \neq \varnothing$ **then**

  return $\ell \equiv f_2(x, y) = \dot{y}_{\min} + \frac{r_y}{2}$;

else if $f_2(Q_{\max}) \cap [\dot{y}_{\max} - \Delta_0, \dot{y}_{\max}] \neq \varnothing$ **then**

  return $\ell ine(b_{\Delta_2}, d_{\Delta_2})$;

else if $f_2(Q_{\max}) \cap [\dot{y}_{\min}, \dot{y}_{\min} + \Delta_0] \neq \varnothing$ **then**

  return $\ell ine(a_{\Delta_1}, c_{\Delta_1})$;

else if $\Delta_1 > \Delta_2$ **then**

  return $\ell ine(a_{\Delta_1}, c_{\Delta_1})$;

else

  return $\ell ine(b_{\Delta_2}, d_{\Delta_2})$;

end

Example On Fig. 4 is shown the invariant of location ℓ_1 of the shared gas-burner, with the position of the level sets of $f_1(x_1, x_2) = h_1 - a_1 x_1 + b_1 x_2$ and $f_2(x_1, x_2) = -a_2 x_2 + b_2 x_1$ at the end of Algorithm 2. The four arrows indicates the moving direction of the lines corresponding to increase the parameter Δ in the algorithm. The shaded regions corresponds to the pieces that are to be separated. The optimal cut is the dashed line $x_1 = x_2$.

4 Refinement-based Safety Verification Algorithm

In this section, we explain the methodology to obtain automatically successive refinements of an affine hybrid automaton H for which to check emptiness.

A first rectangular phase-portrait approximation $H_0 = \mathsf{rect}(H)$ is computed from the original automaton H. Then the automaton H_0 is symbolically analyzed, both forward and backward as described in Section 2. This gives the two sets $\mathsf{Reach}(\llbracket H_0 \rrbracket)$ and $\mathsf{Reach}^{-1}(\llbracket H_0 \rrbracket)$. If their intersection $\mathsf{Unsafe}(\llbracket H_0 \rrbracket)$ is empty, then so is the set $\mathsf{Unsafe}(\llbracket H \rrbracket)$ (by Lemma 7 and Theorem 8) and the emptiness of H is established. Otherwise, we refine the automaton H by splitting one of its unsafe locations and restart the procedure. In fact, from Theorem 25 below and Lemma 10, it appears that refining the rectangular approximation of an automaton H in a safe location is useless for checking emptiness, since the emptiness problem has the same answer for H and its refinement in that case. In other words, the relevant part of the state-space to be refined is $\mathsf{Unsafe}(\llbracket H \rrbracket)$. This is stated by Corollary 26, using the notion of *pruning*.

For an hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$ and a subset $L \subseteq \mathsf{Loc}$ of its locations, let $\mathsf{prune}(H, L)$ be the hybrid automaton $\langle \mathsf{Loc}', \mathsf{Lab}, \mathsf{Edg}', X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$ where $\mathsf{Loc}' = \mathsf{Loc} \backslash L$, $\mathsf{Edg}' = \{(\ell, \sigma, \ell') \in \mathsf{Edg} \mid \ell, \ell' \in \mathsf{Loc}'\}$ and the other components are left unchanged.

Theorem 25 *For every hybrid automaton H, for every subset $L \subseteq \mathsf{SafeLoc}(H)$ of its safe locations, we have $\llbracket \mathsf{prune}(H, L) \rrbracket \approx_{unsafe} \llbracket H \rrbracket$, that is, $\llbracket \mathsf{prune}(H, L) \rrbracket$ and $\llbracket H \rrbracket$ are weakly bisimilar for the unsafe behaviours.*

Corollary 26 *For every hybrid automaton H, for every location $\ell \in \mathsf{SafeLoc}(\mathsf{rect}(H))$, and every hyperplane π, $\mathsf{rect}(H)$ is empty iff $\mathsf{rect}(\mathsf{split}(H, \ell, \pi))$ is empty.*

The core of the refinement based verification procedure is given below. Although the refinement loop is not guaranteed to terminate, we could stop when the size of the invariants run below a certain threshold with the conclusion that the system is not robustly correct for that threshold. In a variation of this procedure, the splitting can be iterated a fixed number of times in each loop, between successive analysis of the rectangularized system.

while $\mathsf{Unsafe}(\llbracket \mathsf{rect}(H) \rrbracket) \neq \emptyset$ **do**
  $L \leftarrow \mathsf{SafeLoc}(\mathsf{rect}(H))$;
  $H' \leftarrow \mathsf{prune}(H, L)$;
  Let ℓ be a location of H' and π be a hyperplane ;
  $H \leftarrow \mathsf{split}(H', \ell, \pi)$;

The splitting is done in the location ℓ having the greatest imprecision (the largest value of sizeRange on its invariant) and the hyperplane π is determined using one of the algorithm presented in Section 3. This is a natural choice for ℓ since our goal is to finally reduce the overall imprecision of the rectangular approximation. This way, the approximation can be made arbitrarily close to the original system (for an infinite-norm metric [10]) and so if the system is robust (in the sense that it is still correct under some small perturbation [6]), our procedure

eventually establishes its correctness, provided the reachability analysis of the rectangular automata terminate. This contrasts with the counter-example based refinement abstraction method (CEGAR) developed by Clarke et al. [4] where the approximations are finite-state, but the refinement procedure is driven by the elimination of spurious counter-examples (executions of the approximation which have no concrete counterpart) and therefore not guaranteed to terminate.

5 Case Study

In practice, we use PPL (the Parma Polyhedra Library [3]) to overapproximate the differential equations by rectangular inclusions. PPL is a C++ library to manipulate polyhedrons with large rational coefficients and exact arithmetic. We analyze the rectangular system with PHAVer [7], a recent tool for the verification of hybrid systems. This tool is designed to verify affine dynamics hybrid automata, based on forward reachability analysis and user-defined rectangular approximations. In our case, since the successive rectangular approximations are obtained automatically, we have disabled the refinement features of PHAVer. Also, we use the reverse automaton construction of Section 2 to implement backward analysis.

We applied our methodology to the Navigation benchmark [5]. We present it briefly. An object is moving on a $m \times n$ grid divided in $m \cdot n$ *cells*. A map M associates to each cell either an integer i in $\{0, \ldots, 7\}$ or a special symbol in $\{\mathsf{A}, \mathsf{B}\}$. The integer determines the *desired velocity* $\mathbf{v}_d(i) = (sin(i\pi/4), cos(i\pi/4))$ in the cell. Then the behaviour of the object (described by its position $\mathbf{x}$ and velocity $\mathbf{v}$) in such a cell is governed by the differential equations $\dot{\mathbf{x}} = \mathbf{v}$ and $\dot{\mathbf{v}} = A \cdot (\mathbf{v} - \mathbf{v}_d(i))$ where A is a 2×2 matrix such that the solution velocity $\mathbf{v}$ always converge to the desired velocity $\mathbf{v}_d(i)$. A cell mapped to B should be avoided and a cell mapped to A should be reached. We only verify the avoidance property and we assume that the object cannot leave the grid. An instance of the benchmark is characterized by the size of the grid, the map M, the matrix A and the initial positions and velocities.

This navigation benchmark has 4 variables x_1, x_2, v_1, v_2 but only two of them (v_1 and v_2) appear in the right-hand side of the differential equations. So, the quality of the splitting is not influenced by the position variables x_1, x_2. Thus it is sufficient to consider line cuts in the plane $v_1 v_2$ of the velocities. In order to apply the procedure $\mathsf{OptimalCut}$, we still have to choose two functions among the four defining the system of differential equations: v_1, v_2, $A_{1*} \cdot (\mathbf{v} - \mathbf{v}_d)$ or $A_{2*} \cdot (\mathbf{v} - \mathbf{v}_d(i))$ where A_{1*} (resp. A_{2*}) is the first (resp. second) row of A. In practice, the first two functions give better results, essentially due to a lower need in computational resources (mainly memory).

The results reported in Fig. 5 are encouraging since we were able to verify more efficiently than PHAVer itself the instances NAV01-04, while we used that tool as a black box for analysis of rectangular automata (thus with all heuristics disabled). Also notice that the instance NAV04 was solved by PHAVer after applying heuristic convex hull and bounding box approximations at particular moments of the analysis [7]. Our results are fragile, since another choice of affine functions has lead to computational difficulties, as mentioned above.

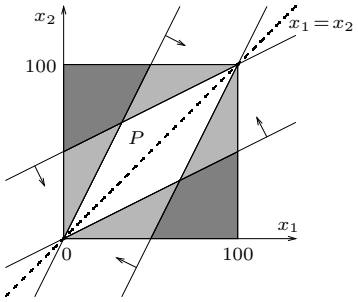


Fig. 4. The optimal cut of the invariant of location ℓ_1 of the shared gas-burner.

Instance	Grid	Time	(PT)
NAV01	3×3	5s	(35s)
NAV02	3×3	10s	(62s)
NAV03	3×3	10s	(62s)
NAV04	3×3	75s	(225s[i])
NAV07	4×4	11mn	

[i] obtained with a heuristic.

Fig. 5. Execution times on a Xeon 3GHz with 4GB RAM (compared with PT = PHAVER times [7]).

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
2. Eugene Asarin, Thao Dang, Oded Maler, and Olivier Bournez. Approximate reachability analysis of piecewise-linear dynamical systems. In *Proceedings of HSCC 2000: Hybrid Systems, Computation and Control*, LNCS 1790, pages 20–31. Springer, 2000.
3. R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In *Static Analysis: Proceedings of the 9th International Symposium*, LNCS 2477, pages 213–229, Springer, 2002.
4. Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proceedings of CAV 2000: Computer Aided Verification*, LNCS 1855, pages 154–169. Springer, 2000.
5. Ansgar Fehnker and Franjo Ivancic. Benchmarks for hybrid systems verification. In *HSCC 04: Hybrid Systems, Computation and Control*, LNCS 2993, pages 326–341. Springer, 2004.
6. Martin Fränzle. Analysis of hybrid systems: An ounce of realism can save an infinity of states. In *Proceedings of CSL 1999: Computer Science Logic*, LNCS 1683, pages 126–140. Springer, 1999.
7. Goran Frehse. Phaver: Algorithmic verification of hybrid systems past HyTech. In *HSCC 05: Hybrid Systems, Computation and Control*, LNCS 3414, pages 258–273. Springer, 2005.
8. T.A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society Press, 1996.
9. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HyTech. In *TACAS 95: Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1019, pages 41–71. Springer, 1995.
10. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.
11. T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.
12. R. Milner. *A Calculus of Communicating Systems*. LNCS 92. Springer, 1980.