Deriving Liveness Properties of Hybrid Systems from Reachable Sets and Lyapunov-like Certificates

Ludovico Battista[0000−0001−8197−2001] and Stefano Tonetta[0000−0001−9091−7899]

Fondazione Bruno Kessler, Trento 38123, Italy

Abstract. In this paper we tackle the problem of proving generic LTL properties of hybrid systems with a particular focus on liveness properties such as recurrence of regions, response to stimuli, or region stability. Although many advances have been made in the analysis of reachability and safety properties of dynamic and hybrid systems, there is a lack of tool support for the automated verification of liveness properties. We propose a fully automated approach that combines Lyapunov synthesis, reachability analysis and SMT-based quantifier elimination to build a discrete abstraction and then applies standard model checking algorithms. A key step of the algorithm is the derivation of LTL constraints from reachable sets computations and Lyapunov-like certificates. We implemented the approach on top of the CORA and nuXmv tools and show how it can scale in the size of the discrete structure and how it can prove properties over linear and non-linear complex dynamics.

Keywords: Hybrid Systems · Linear Temporal Logic · Discrete Abstraction · Lyapunov Functions · Certificate Functions · Reachable Sets Overapproximation.

1 Introduction

The verification of temporal properties in hybrid systems [Hen96,ACH$^+$95,GST12] poses significant challenges due the interaction between continuous dynamics and discrete transitions. This problem is further exacerbated when considering properties of reactive systems that need to hold on infinite executions, and in particular liveness properties such as recurrence of regions, response to stimuli, and region stability.

These properties can be expressed in LTL [Pnu77], but while model-checking techniques have been successful in verifying LTL on discrete-time transition systems, most approaches to the verification of hybrid systems focus on safety and reachability properties (cfr., e.g., [SÁW$^+$24] for a recent overview of the state-of-the-art technology). Although a number of works also proposed techniques for LTL (e.g., [PKV13,CGMT14,HS20]), they are typically limited either in the expressiveness of the dynamics and the properties or in the automation and scalability of the verification.

Object	**Property**	**Type**
Lyapunov Function	$G(F(G(\neg R) \vee (\neg q)))$	Liveness
Descent Function	$G(F(G(\neg R) \vee (\neg q)))$	Liveness
Barrier Function	$G(R_1 \rightarrow (G(\neg R_2)W(\neg q)))$	Safety
Reachable Set Overapproximation	$R_1 \wedge (R_1 U(R_2 \wedge R_2 U(R_3 \wedge R_3 U \ldots)))$	Liveness

Table 1. Table that summarizes the form of LTL properties generated from each analysis technique. R_i are regions expressed as Boolean combination of polynomial constraints and q represents a location of the hybrid automaton.

In the direction of closing such gap, we propose a new algorithm, called `LTLConstraints4HS`, to prove that a hybrid system H satisfies an LTL formula ϕ based on the following steps steps: 1) generate a set of regions from the property to prove, sublevel sets of Lyapunov functions, and post-images through discrete jumps, 2) derive a set Γ of LTL properties that are satisfied by H from certificate functions and from Reachability Set Overapproximation (RSO) applied to the regions, 3) build a discrete-time coarse overappoximation D_H of the hybrid system based on the regions, 4) call a model checker to prove $D_H \models (\bigwedge_{\gamma \in \Gamma} \gamma) \rightarrow \phi$. The core of the method is the second step, which exploits existing analysis techniques such as Lyapunov-like functions, barrier certificates and reachability analysis to generate LTL properties that are satisfied by H. Such generation is focused on liveness properties as summarized in the Table 1.

We implemented the approach on top of the CORA [Alt15] and nuXmv [CCD$^+$14] tools and show the effectiveness of the method both in handling systems with large discrete structures, and in proving properties over linear and non-linear complex dynamics. The method is fully automated but not complete (the problem is in general undecidable).

The precision of the abstraction depends on the set of regions and we allow to specify additional regions in input to the algorithm. Nevertheless, we are able to automatically prove many interesting properties on complex models without any additional regions. No other tool is currently able to automatically tackle such a task. The experimental results show that `LTLConstraints4HS` can generate and handle up to $\sim$ 150 regions and $\sim$ 1000 LTL constraints. Code and complete results are available at [BT].

Our contributions may be summarized as follows:

- a new general method that integrates standard techniques for hybrid dynamical systems with LTL discrete-time model checking;
- a method to synthesize regions for the discrete abstraction that is empirically proven to be effective in several benchmarks;
- derivation of LTL properties from certificate functions and RSO;
- an implementation of the method that automates the four steps of synthesizing certificate functions and regions, generating the LTL properties, abstracting, and model checking the result;
- an experimental evaluation that shows the effectiveness of the approach on new or existing benchmarks ([FI04,BKRS24]).

Acknowledgments We acknowledge the support of the MUR PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU.

Related Work Various abstraction techniques have been proposed for hybrid systems [AGRS25], typically with some assumptions on the dynamics. For linear hybrid systems, predicate abstraction and CEGAR have been proved to be effective for verifying safety properties [ADI06]; liveness-related properties have been proved under some conditions on equilibrium points [CN12]; complete discrete abstractions have been found if the continuous or discrete evolution are restricted to specific types [AHLP00]; SMT-based model checking has been applied to rectangular hybrid automata [CGMT14]. Our approach builds a rather simple abstraction, but potentially tackles arbitrary polynomial systems.

Other works focus on the use of certificate functions, but usually for safety and reachability analysis. Lyapunov-like functions have been considered to build sublevel sets to create an abstract timed automaton [WS11] or generalized to recover ideas similar to k-induction [Bak18], [AMTZ21]. Contrary to these works, here we use Lyapunov-like functions in combination with other methods to derive general liveness properties for the systems.

Reachability analysis [Alt10], [AFG21] is a powerful way to study properties of hybrid systems, and several tools are devoted to this task [dC16], [CBGV12], [BFF$^+$19], [Alt15]. They efficiently prove safety properties for bounded times and have been used to approach unbounded time safety properties for specific systems [PA18]. Although we make use of several tools and approaches that were already available (*e.g.* numerical solvers to compute Lyapunov-like functions, CORA to compute RSO, SMT-solvers to build the discrete abstraction and to check the validity of the generated certificate functions, nuXmv for the model checking), we are able to provide a framework that integrates all of them to successfully prove LTL properties on hybrid systems.

Some works have approached the verification of liveness properties, using Lyapunov-like functions as certificates for the validity of temporal properties [HS20] [NEDR$^+$23], and, in the context of non-linear systems, to invalidate substrings to prove general LTL properties implied by safety properties [WTL14]. KeYmaera X [FMQ$^+$15] provides an excellent framework for proving temporal properties also using certificate functions. Compared to these, we provide an automated method that combines regions and certificates synthesis, RSO, and discrete-time symbolic model checking in a unique novel way.

2 Background

2.1 Finite-state Transition Systems and LTL

Given a finite set V of Boolean variables, let V' denote a copy of the variables V, which are used to represent the values of V after a transition. A *Finite-state Transition System* (FTS) S is a tuple $S = \langle V, \mathit{Init}, \mathit{Inv}, \mathit{Trans} \rangle$, where V is a set of (state) variables, *Init* is a formula over V representing the initial condition, *Inv* is a formula over V representing the invariant condition, and *Trans* is a formula over $V \cup V'$ representing the transition condition. A state $s \subseteq 2^V$ of S is a Boolean assignment to the variables V, represented as a subset of V.

A trace σ of S is an infinite sequence of states $\sigma = s_0, s_1, \cdots$ such that $s_0 \models \mathit{Init}$ and for all $i \geq 0$, $s_i \models \mathit{Inv}$, and $s_i \cup s'_{i+1} \models \mathit{Trans}$.

LTL We define the set of LTL formulas over the variables V with the following grammar rule: $\phi := p \mid \phi \vee \phi \mid \neg\phi \mid \phi U \phi$ where $p \in V$. Note that we do not consider here the standard next (X) operator. We use the following standard abbreviations: $\top := p \vee \neg p$, $\bot := \neg\top$, $\phi \wedge \psi := \neg(\neg\phi \vee \neg\psi)$, $\phi \rightarrow \psi := (\neg\phi) \vee \psi$, $F\phi := \top U \phi$, $G\phi := \neg F \neg\phi$, $\psi W \phi := (\psi U \phi) \vee G(\psi)$. Traces over V are infinite sequences of assignments to V. Given a trace $\sigma = s_0, s_1, \ldots$, we denote with $\sigma[i]$ the $i+1$-th state s_i and with σ^i the suffix trace starting from $s[i]$. Given a trace σ and an LTL formula ϕ over V, we define $\sigma \models \phi$ in the usual way as follows: $\sigma \models p$ iff p evaluates to true given the assignment $\sigma[0]$; $\sigma \models \phi \vee \psi$ iff $\sigma \models \phi$ or $\sigma \models \psi$; $\sigma \models \neg\phi$ iff $\sigma \not\models \phi$; $\sigma \models \phi U \psi$ iff there exists $i \geq 0$ s.t. $\sigma^i \models \psi$ and for all j, $0 \leq j < i$, $\sigma^j \models \phi$. Given an FTS S and an LTL formula ϕ over V, $S \models \phi$ if for all traces σ of S, $\sigma \models \phi$.

2.2 Hybrid Systems

Several formal models fall under the name of hybrid systems. Here, we consider the classical definition of hybrid automaton [Hen96]:

Definition 1. *A* Hybrid Automaton *(HA) is a tuple* $H = (X, Q, q_{init}, E, init, flow, inv, guard, jump)$, *where:*
- X *is a finite set of real variables* $x_1, \ldots, x_n$*;*
- Q *is a finite set of discrete modes (also called locations);*
- $q_{init} \in Q$ *is the initial location;*
- $E \subseteq Q \times Q$ *is the set of possible discrete transitions;*
- $init \subseteq \mathbb{R}^n$ *specifies the set of initial continuous states;*
- $inv \colon Q \rightarrow 2^{\mathbb{R}^n}$ *assigns an invariant set* $inv(q)$ *to each location* q*, that specifies where the system is allowed to remain while in mode* q*;*
- $flow \colon Q \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ *is a partial function that defines the continuous evolution of the system in each discrete mode* $q \in Q$ *via a set of differential equations;*
- $guard \colon E \rightarrow 2^{\mathbb{R}^n}$ *is a guard condition, specifying when the transition* (q, q') *is enabled based on the continuous state;*
- $jump \colon E \rightarrow 2^{\mathbb{R}^n \times \mathbb{R}^n}$ *assigns a relation to each edge that specifies the possible* jumps *of the continuous variables.*

A state of H is a pair $\langle q, s \rangle$, where $q \in Q$ and s is an assignment to the variables in X; s is also called a continuous state.

We consider a symbolic representation of the conditions $init$, $inv(q)$, $flow(q, s)$, $guard(e)$, and $jump(e)$, in the sense that they are defined by symbolic formulas over the variables X, taken from a set denoted by $Expr(X)$ (or $Expr(X, X')$ in the case of $jump$). In this work, we focus on *polynomial automata*, where

$Expr(X)$ contains Boolean combinations of polynomial constraints over X. This is mainly due to limitations of theory solvers and simulators, but the proposed method can be generalized to more general forms. We also assume, for simplicity, that E does not contain elements of the form (q, q).

To describe the evolution of a hybrid system H we will use the notion of *hybrid trace* [dAM95,CRT09]. We briefly recall it. Given an interval $I \subseteq \mathbb{R}$, we denote by $l(I)$ its infimum and by $u(I)$ its supremum.

Definition 2. *A hybrid trace for H is an infinite sequence $\sigma = \langle f_0, I_0, q_0\rangle, \langle f_1, I_1, q_1\rangle, \ldots$ such that: I_i are adjacent intervals,* i.e. *$l(I_{i+1}) = u(I_i)$; q_i are locations,* i.e. *$q_i \in Q$; the intervals cover $\mathbb{R}^{\geq 0}$,* i.e. *$\bigcup_{i\in\mathbb{N}} I_i = \mathbb{R}^{\geq 0}$; $f_i \colon I_i \to \mathbb{R}^n$ is analytic. We sometimes denote q_i with $loc(\sigma_i)$.*

Definition 3. *A hybrid trace is a* trajectory *(also called a* run *or a* solution*) for H if:*
- *the image of f_i is contained in $inv(q_i)$.*
- *if $l(I_i) \neq u(I_i)$, then $\frac{df_i}{dt}(t) = flow(q_i)(f_i(t))$ for all $t \in I_i$;*
- *if $q_i = q_{i+1}$, then $f_i \cup f_{i+1}$ is well defined and analytic on $I_i \cup I_{i+1}$. In particular, either I_i is right-closed or I_{i+1} is left-closed;*
- *if $q_i \neq q_{i+1}$, then I_i is right-closed, I_{i+1} is left-closed, $e = (q_i, q_{i+1}) \in E$, $f_i(u(I_i)) \in guard(e)$, and $(f_i(u(I_i)), f_{i+1}(l(I_{i+1}))) \models jump(e)$.*

We say that a run *starts in* (R, q) if $f_0(0) \in R$ and $loc(\sigma_0) = q$. An *initial run* is a run that starts in $(init, q_{init})$. We notice that different traces may describe the same dynamics. Intuitively, a trace is a sampling refinement of another one if it has been obtained by splitting an interval into two parts [dAM95].

Definition 4 (Partitioning Function - Sampling Refinement [dAM95]). *A partitioning function μ is a sequence $\mu_0, \mu_1, \mu_2, \ldots$ of non-empty, adjacent and disjoint intervals of $\mathbb{N}$ partitioning $\mathbb{N}$. Formally, $\bigcup_{i\in\mathbb{N}} \mu_i = \mathbb{N}$ and $u(\mu_i) = l(\mu_{i+1}) - 1$.*

Given two hybrid traces $\sigma = \langle f_0, I_0, q_0\rangle, \langle f_1, I_1, q_1\rangle, \ldots$ and $\sigma' = \langle f'_0, I'_0, q'_0\rangle, \langle f'_1, I'_1, q'_1\rangle, \ldots$, we say that σ' is a sampling refinement of σ by the partitioning μ (denoted by $\sigma' \preceq_\mu \sigma$) iff, for all $i \in \mathbb{N}$, $I_i = \bigcup_{j\in\mu_i} I'_j$, and for all $j \in \mu_i$, $f'_j = f_i$ and $q'_j = q_i$.

2.3 LTL on Hybrid Traces (HLTL)

We extend LTL replacing propositions with predicates over a set Q of locations and a set X of real variables. To distinguish it from the propositional version defined in Section 2.1, we refer to this extension as HLTL, whose syntax is given by the following grammar rules: $\phi := q \mid R \mid \phi \vee \phi \mid \neg\phi \mid \phi U \phi$, where $q \in Q$ and R ranges over $Expr(X)$. We use the same abbreviations defined for LTL.

Given a hybrid trace $\sigma = \langle f_0, I_0, q_0\rangle, \langle f_1, I_1, q_1\rangle, \ldots$ and an HLTL formula ϕ over Q and X, we define $\sigma \models \phi$ as follows:
- $\sigma, i \models q$ iff $q_i = q$;
- $\sigma, i \models R$ iff for all $t \in I_i$, $f_i(t) \models R$;

- $\sigma, i \models \phi \vee \psi$ iff $\sigma, i \models \phi$ or $\sigma, i \models \psi$
- $\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$
- $\sigma, i \models \phi U \psi$ iff there exists $k \geq i$ s.t. $\sigma, k \models \psi$ and for all j, $i \leq j < k$, $\sigma, j \models \phi$

Finally, $\sigma \models \phi$ iff $\sigma, 0 \models \phi$. We say that a hybrid trace σ is ground for a a predicate R if its interpretation over intervals is constant, *i.e.*, $\forall (t_1, t_2) \in I_i$, $(f_i(t_1) \models R \Leftrightarrow f_i(t_2) \models R)$. We say that σ is ground for an HLTL formula ϕ iff it is ground for every predicate in ϕ. Given a hybrid automaton H and an HLTL formula ϕ over Q and X, $H \models \phi$ iff, for all initial runs σ of H that are ground for ϕ, $\sigma \models \phi$. Notice that it is always possible to refine a run to make it ground for a HLTL formula [dAM95,CRT09].

Definition 5. *We sometimes identify a formula $R \in Expr(X)$ with the set of states that satisfy it,* i.e., $\{s \in \mathbb{R}^n \ | s \models R\}$. *We call both the formula and the associated set a* region *of H.*

We denote by $\overline{R}$ the formal closure *of R, obtained by performing the following steps: 1) rewrite R into DNF; 2) remove negations in front of the polynomial constraints by interchanging $<$ with $\geq$, $>$ with $\leq$, $=$ with $\neq$; 3) apply the substitutions $a > b \Rightarrow a \geq b$, $a < b \Rightarrow a \leq b$, $a \neq b \Rightarrow \top$. Note that this is not the topological closure $clos(R)$ of R. In general, it holds that $\overline{R} \supseteq clos(R)$.*

Definition 6. *Predicates of the form $R_Z \wedge (\bigvee_{q \in Q_Z} q)$ are often very natural and useful. These corresponds to the combination of a region $R_Z \in Expr(X)$ and a set of locations $Q_Z \subseteq Q$. We support them natively:* a local region *Z is a pair (R_Z, Q_Z). Notice that a region is logically equivalent to a particular case of local region, where $Q_Z = Q$. For this reason, we will only consider local regions in the main algorithm. A local region Z in a HLTL predicate is interpreted as $R_Z \wedge (\bigvee_{q \in Q_Z} q)$.*

2.4 Lyapunov-like, Descent, Barrier functions

Certificate functions are used to ensure the satisfaction of some properties such as Lyapunov-like functions to prove stability or Barrier functions to prove safety properties.

Lyapunov-like function Even when a hybrid system does not converge to a single attracting point, it can admit functions that decrease along trajectories. To better exploit this property, we give the following definition of Lyapunov-like function [Kel14,Bra94] using class-κ functions [Kha02]:

Definition 7. *A couple $L = (L_f, L_Q)$ where L_Q is a subset of Q and L_f is a function $L_f \colon \mathbb{R}^n \times L_Q \to \mathbb{R}$; is a* Lyapunov-like function for *H if:*
- *for every $q \in L_Q$, and for every (x, q) such that $x \in inv(q)$, we have $L_f(x, q) \geq 0$, and the Lie derivative is non-positive: $\mathcal{L}_{flow(q)} L_f(x, q) \leq 0$;*
- *for every edge $(q, q') \in E$ such that $\{q, q'\} \subseteq L_Q$, $x \in guard(e)$ and $(x, x') \models jump(e)$, we have $L_f(x', q') \leq L_f(x, q)$;*

– *there exists a κ-function α such that for every $q \in L_Q$ and $x \in inv(q)$, we have $\mathcal{L}_{flow(q)} L_f(x, q) \leq -\alpha(L_f(x, q))$.*

Informally, L_f is the value of the Lyapunov-like function in the different locations and L_Q is the set of locations where the Lyapunov-like conditions hold. We will generalize results such as the Comparison Principle [Kel14, Section 5] to our setting in Section 3.3.

Descent function An analogue tool that proved to be useful is:

Definition 8. *A couple $D = (D_f, D_Q)$ where D_Q is a subset of Q and D_f is a function $D_f : \mathbb{R}^n \times Q \to \mathbb{R}$ is a* Descent function for H *if there is $\varepsilon > 0$ such that*

– *for every $q \in D_Q$, if $x \in inv(q)$, we have $\mathcal{L}_{flow(q)} D_f(x, q) < -\varepsilon$;*
– *for every edge $(q, q') \in E$ such that $\{q, q'\} \subseteq D_Q$, $x \in guard(e)$ and $(x, x') \models jump(e)$, we have $D_f(x', q') \leq D_f(x, q)$*

Informally, D_f is a function that decreases indefinitely over trajectories that stay in the locations D_Q.

Barrier function Another important tool that we use is the following:

Definition 9. *A couple $B = (B_f, B_Q)$ where B_Q is a subset of Q and B_f is a function $B_f : \mathbb{R}^n \times Q \to \mathbb{R}$ is a* Barrier function for H *if:*

– *for every $q \in B_Q$, if $x \in inv(q)$ and $B_f(x, q) = 0$, $\mathcal{L}_{flow(q)} B_f(x, q) < 0$;*
– *for every edge $(q, q') \in E$ such that $\{q, q'\} \subseteq B_Q$, $x \in guard(e)$ and $(x, x') \models jump(e)$, we have $B_f(x, q) \leq 0 \to B_f(x', q') \leq 0$.*

Informally, B_f provides a certificate that a trajectory will not get from $B_f \leq 0$ to $B_f > 0$ while staying in locations B_Q.

Definition 10. *In the context of this paper, a certificate function is either a Lyapunov-like, or a descent, or a barrier function.*

Notation 1 *With abuse of notation, we sometimes denote with $L_f(q)$ the restriction of L_f to the set $\mathbb{R}^n \times \{q\}$. This also reflects the implementation, where $L_f(q)$ is represented by a polynomial expression. We do the same for descent and barrier functions.*

Methods for Synthesis of Certificate Functions Many numerical methods exist to synthesize certificate functions for continuous systems [Par00,ZZCL20], and several variations had been proposed more recently to work with hybrid systems [BT24,Oeh11]. Several works are also devoted to the formal verification of the correctness of such functions [APA20]. We support the synthesis of candidate certificate functions through: 1) the use of Sum of Squares (SOS) programming, followed by a soundness check through an SMT solver; 2) directly using an SMT solver to synthesize a valid function (see *e.g.* [APA20]). We elaborate on this in Section 3.5.

Notice that these approaches are by no means exhaustive: synthesizing certificate functions (in particular for non-linear systems) is an active area of research. One important aspect of our framework is that it can be easily extended to support any method for the synthesis of certificate functions.

2.5 Reachable Set Overapproximation (RSO)

Tools that provide reachable sets ([GG08]) using RSO form a powerful way to obtain information on the behaviour of a hybrid system.

Definition 11. *Given a hybrid system H, a starting region R, and a starting location q, a* Reachable Set Overapproximation (or RSO) with granularity τ up to time T ($= \tau N$ for some $N \in \mathbb{N}$) *for (R, q) is a finite set $\mathcal{B} = \{(Y_i, J_i, p_i)\}$ where: Y_i is a region of H, $J_i = [l_i\tau, u_i\tau]$ is an interval, and $p_i \in Q$, such that for every run $\sigma = \langle f_j, I_j, q_j\rangle$ with $f_0(0) \in R$ and $q_0 = q$, for every $0 \le t \le T$ and every j such that $t \in \operatorname{Dom} f_j$, the point $f_j(t)$ is inside $\bigcup_{i \mid t \in J_i, p_i = q_j} Y_i$.*

Informally, a RSO gives information on where a solution that starts in the region R and location q can be at any time $0 \le t \le T$.

3 Discrete Abstraction with LTL Constraints

3.1 Overview

In this section, we detail the steps of Algorithm `LTLConstraints4HS`, that we use to verify the validity of an HLTL property for a hybrid system H, using a discrete abstraction and LTL constraints. The algorithm takes as input:
- a hybrid system H as defined in Def. 1;
- a set $\mathcal{Z}_\phi$ of local regions $Z = (R_Z, Q_Z)$ (as defined in Def. 6) that are used to define the HLTL property ϕ, and a (possibly empty) set $\mathcal{Z}_{add}$ of additional regions to be used in the abstraction. Let $\mathcal{Z}_{in} := \{(init, q_{init})\} \cup \mathcal{Z}_\phi \cup \mathcal{Z}_{add}$;
- an HLTL property ϕ defined over Q and $\mathcal{Z}_\phi$.

Algorithm `LTLConstraints4HS` consists of the following steps:

1. it synthesizes and validates Lyapunov-like and descent functions;
2. it synthesizes local regions $\mathcal{Z}_{Lyap}$, defined as some sublevels sets of Lyapunov-like functions and possibly their images through admissible *jump* functions; let $\mathcal{Z} = \mathcal{Z}_{in} \cup \mathcal{Z}_{Lyap}$;
3. it computes the barrier functions corresponding to the local regions $\mathcal{Z}_{Lyap}$; let $\mathcal{F}$ be all the certificate functions found;
4. it computes an HLTL formula γ_f over $\mathcal{Z}$ and Q for each $f \in \mathcal{F}$;
5. it performs RSO for each local region $(R_Z, Q_Z) \in \mathcal{Z}$ and each $q \in Q_Z$ and computes the HLTL formula $\gamma_{Z,q}$ over $\mathcal{Z}$ and Q (the granularity τ and time T are fixed parameters of the algorithm).
6. it computes the discrete abstraction D_H, which has a variable v_Z for each local region $(R_Z, Q_Z) \in \mathcal{Z}$ and a variable v_q for each location $q \in Q$;

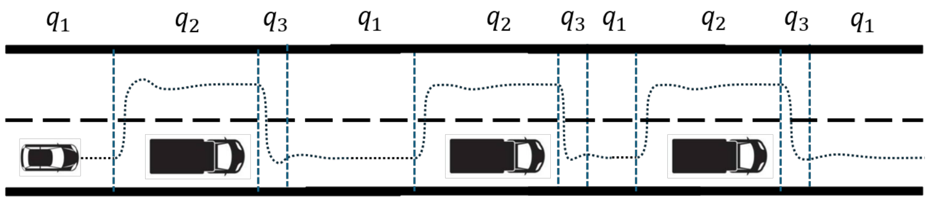

Fig. 1. Simple example of a car overtaking obstacles on the right lane of a highway.

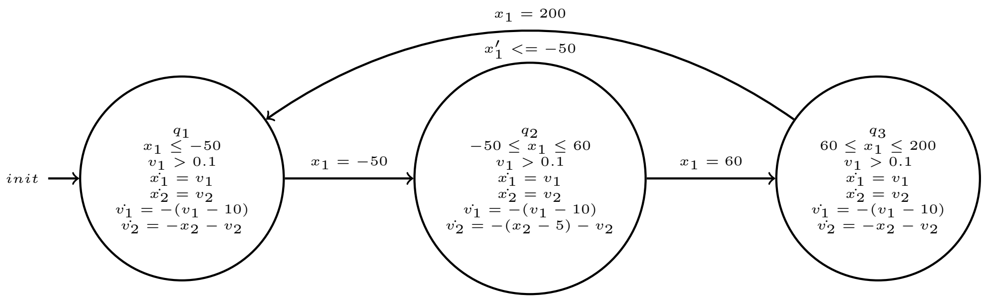

Fig. 2. Hybrid automaton for the car example.

7. it transforms ϕ and each generated γ_i into LTL properties $\widehat{\phi}$ and $\widehat{\gamma_i}$ substituting Z with v_Z and q with v_q;
8. if $D_H \models (\bigwedge_{f \in \mathcal{F}} \widehat{\gamma_f} \wedge \bigwedge_{Z \in \mathcal{Z}, q \in Q_Z} \widehat{\gamma_{Z,q}}) \rightarrow \widehat{\phi}$, then it returns $True$;
9. else it returns $Unknown$.

We detail steps 4-5-6 in the next sections, postponing steps 1-2-3 to Section 3.5 (since it contains more standard arguments). We start by defining an example that will help us in the description of the algorithm.

3.2 Running example

We present a simple hybrid system model to exemplify the method. The model represents a car running on a two-lane highway, overtaking obstacles on the road, like depicted in Fig. 1. For simplicity, the obstacles are always on the right lane and not moving. The movement of the car is divided in three phases that are repeated in sequence infinitely often: q_1 where the car approaches the obstacle, q_2 where it changes lane, q_3 where it returns to the original lane after overtaking the obstacle. After the overtake (q_3) the car returns to q_1 to approach another obstacle.

The hybrid automaton formalizing the example is shown in Fig. 2, where x_1, x_2, v_1, v_2 are the continuous variables representing the distance from the engaged obstacle x_1, the distance from the center of the right lane x_2, and the velocity (v_1, v_2) of the vehicle; the initial region is $init$ defined as $x_1 \leq -100 \wedge -0.1 \leq$

$x_2 \leq 0.1 \wedge 9.9 \leq v_1 \leq 10.1 \wedge -0.1 < v_2 < 0.1$, and every jump condition is abbreviated so that if a variable is not mentioned is implicitly not changed.

The local regions $\mathcal{Z}_\phi$ are those that we want to avoid: the part beyond the right lane $B_1 = (x_2 < -2, \{q_1, q_2, q_3\})$, the one beyond the left lane $B_2 = (x_2 > 7, \{q_1, q_2, q_3\})$, and the part surrounding the engaged obstacle $B_3 = (0 \leq x_1 \leq 50 \wedge -0.5 \leq x_2 \leq 0.5, \{q_2\})$. We add no other local regions, *i.e.* $\mathcal{Z}_{add} = \emptyset$.

We would like to prove that the car overtakes all obstacles, or in other words that whenever the car has an obstacle in front of it (q_1), it will eventually reach a position after the obstacle (q_3) while avoiding the bad states defined as $B := B_1 \vee B_2 \vee B_3$. This is formalized by the HLTL property

$$\phi := G(q_1 \rightarrow ((\neg(B))Uq_3)).$$

Lyapunov-like and descent functions Using SOS (see Sec. 2.4) we compute (and validate symbolically) the following Lyapunov-like functions:
$L^1 = \Big(L^1_f(x, q_1) = L^1_f(x, q_3) = \kappa_1 v_2^2 + (\kappa_2 v_2 + \kappa_3 x_2)^2 + \kappa_4(v_1 - 10)^2, \{q_1; q_3\}\Big)$;
$L^2 = \Big(L^2_f(x, q_2) = \kappa_5 v_2^2 + (\kappa_6 v_2 + \kappa_7(x_2 - 5))^2 + \kappa_8(v_1 - 10)^2, \{q_2\}\Big)$;
where κ_i are positive constants. Using symbolic methods (see Sec. 2.4) we also compute the following descent functions:
$D^1 = \Big(D^1_f(x, q_1) = D^1_f(x, q_2) = -\kappa_{11} v_1 - \kappa_{12} x_1, \{q_1; q_2\}\Big)$;
$D^2 = \Big(D^2_f(x, q_3) = -\kappa_{11} v_1 - \kappa_{12} x_1, \{q_3\}\Big)$.

Additional local regions and barrier functions We automatically select 9 sublevel sets of each of the two Lyapunov-like function. We add them and the images through the jump functions of their non-empty intersections with guards to the local regions, for a total of $36 = |\mathcal{Z}_{Lyap}|$ regions. For clarity and space reasons, we now list only four of these local regions, that are shown in Figure 3:
$Z_1 := (L^1_f \leq \kappa_9, \{q_1; q_3\})$; $Z_2 := (L^1_f \leq \kappa_9 \wedge x_1 = -50, \{q_2\})$;
$Z_3 := (L^2_f \leq \kappa_{10}, \{q_2\})$; $Z_4 := (L^2_f \leq \kappa_{10} \wedge x_1 = 60, \{q_3\})$;

For each sublevel set we add the associated barrier function. For example, for Z_1 we have: $B^1 = (B^1_f(x, q_1) = B^1_f(x, q_3) = L^1_f - \kappa_9, \{q_1; q_3\})$.

In Section 3.5, we detail the process we follow for the above computation. However, we focus first on how we derive the LTL constraints and the discrete abstraction from these objects.

3.3 LTL constraints

In this section we are going to define HLTL formulae that hold for the system H. We will then use that $H \models \gamma$ and that $D_H \models (\widehat{\gamma} \rightarrow \widehat{\psi})$ to conclude $H \models \psi$. The complete proofs are available in the Appendix, Section B.

Certificate functions We start by listing HLTL properties that can be derived from the existence of certificate functions (recall Definition 10), together with the intuition behind their construction.

Lyapunov-like functions Suppose that $L = (L_f, L_Q)$ is a valid Lyapunov-like function for H. If a solution will eventually be contained in the locations L_Q, the value of L_f along such solution will converge to 0. For this reason, if a local region $Z = (R_Z, Q_Z)$ is such that the infimum of $L_f(q)$ is strictly positive on R_Z for some $q \in Q_Z \cap L_Q$, we know that such a solution will eventually leave R or L_Q. We formalize this intuition in the following theorem:

Theorem 1. *Let $L = (L_f, L_Q)$ be a Lyapunov-like function for H. Let (R_Z, Q_Z), $q \in L_Q \cap Q_Z$ be such that for some $\varepsilon > 0$ the following formula is valid: $((R_Z \wedge inv(q))) \to L_f(q) > \varepsilon$. Then:*
$H \models G\Big(G\Big(\bigvee_{p \in L_Q} p\Big) \to F(G(\neg(Z \wedge q)))\Big)$*, or, equivalently,*
$H \models G\Big(F\Big(G(\neg(Z \wedge q)) \vee \neg\Big(\bigvee_{q \in L_Q} q\Big)\Big)\Big)$.

Given L, for every $q \in L_Q$ and every $Z \in \mathcal{Z}, q \in Q_Z$ we look for the existence of a valid ε for $R_Z \wedge inv(q)$ and $\neg R_Z \wedge inv(q)$. This is done by checking the validity of the formula in the proposition for $\varepsilon = 10^{-4}$. Finally, for every Lyapunov function L we compute γ_L as the conjunction of the HLTL propositions on the pairs (Z, q) for which we are able to prove the existence of such ε. Details can be found in Appendix D.
Running example: Since $L^1_f(q_1)$ has a positive minimum on $\neg R_{Z_1} \wedge inv(q_1)$, we have $H \models G(G(q_1 \vee q_3) \to F(G(\neg(\neg Z_1 \wedge q_1))))$, and similarly for other combinations.

Descent functions The case of descent functions is similar to the one of Lyapunov-like functions. Also in this case, these functions decrease along solutions that are contained in the locations D_Q, and we can deduce that such solution will leave local regions that have a finite infimum of D_f on them. This is formalized in the following theorem:

Theorem 2. *Let $D = (D_f, D_Q)$ be a descent function for H. Let (R_Z, Q_Z), $q \in D_Q \cap Q_Z$ be such that for some $M \in \mathbb{R}$ the following formula is valid: $(R_Z \wedge inv(q)) \to D_f(q) > M$. Then: $H \models G\Big(G\Big(\bigvee_{p \in D_Q} p\Big) \to F(G((\neg(Z \wedge q))))\Big)$.*

We compute γ_D in the same way we did for Lyapunov-like functions.
Running example: Every pair (Z, q) satisfies the hypotheses of Theorem 2 for some D^i. As an example, for the pair (Z_1, q_1) the HLTL proposition is modeled by H: $H \models G(G(q_1 \vee q_2) \to F(G((\neg(Z_1 \wedge q_1)))))$.

Barrier functions Suppose that $B = (B_f, B_Q)$ is a valid barrier function for H. Let Z_1 and Z_2 be two local regions such that, for some locations $q_1, q_2 \in B_Q \cap Q_{Z_1} \cap Q_{Z_2}$, the value of $B(q_1)$ on R_{Z_1} is strictly positive and the value of $B(q_2)$ on R_{Z_2} is negative. Since the barrier function will not pass from negative to positive values along a solution that stays in the locations in B_Q, we know that if such solution gets to R_{Z_2} in location q_2, it will not get to R_{Z_1} in location q_1, unless it passes through locations outside B_Q. This is formalized in the following theorem:

Theorem 3. *Let $B = (B_f, B_Q)$ be a barrier function for H. Let (R_{Z_1}, Q_{Z_1}), (R_{Z_2}, Q_{Z_2}), $q_1 \in B_Q \cap Q_{Z_1}$, $q_2 \in B_Q \cap Q_{Z_2}$, be such that* $((R_{Z_1} \wedge inv(q_1)) \rightarrow B_f(q_1) > 0) \wedge ((R_{Z_2} \wedge inv(q_2)) \rightarrow B_f(q_2) \leq 0)$*, Then:* $H \models G\Big((Z_2 \wedge q_2) \rightarrow \Big((\neg(Z_1 \wedge q_1))W\Big(\neg \bigvee_{p \in B_Q} p\Big)\Big)\Big)$.

Given B, let $(Z_i, q_i)_{i \in I}$ be such that $((R_{Z_i} \wedge inv(q_i)) \rightarrow B_f(q_i) > 0)$ and $(Z_j, q_j)_{j \in J}$ such that $((R_{Z_j} \wedge inv(q_j)) \rightarrow B_f(q_j) \leq 0)$. We define γ_B as the following formula, whose validity is a straightforward consequence of Theorem 3: $\gamma_B := G\Big(\big(\bigvee_{i \in I}(Z_i \wedge q_i)\big) \rightarrow \Big(\big(\bigwedge_{j \in J}(\neg(Z_j \wedge q_j))\big)W\Big(\neg \bigvee_{p \in B_Q} p\Big)\Big)\Big)$.
Running example: Since $(R_{Z_1} \wedge inv(q_1)) \rightarrow B^1_f(q_1) \leq 0$ and $(\neg R_{Z_1} \wedge inv(q_1)) \rightarrow B^1_f(q_1) > 0$, it holds: $H \models G((Z_1 \wedge q_1) \rightarrow ((\neg((\neg Z_1) \wedge q_1))W(\neg(q_1 \vee q_3))))$, and similarly for other combinations.

Reachability matrices from RSO Here, we transform the information contained in a RSO $\mathcal{B} = \langle Y_i, J_i, p_i \rangle$ (Definition 11) in an HLTL property that is satisfied by the system H.

To do this, we start by creating a schematic object that can be easily manipulated, called *reachability matrices*:

Definition 12. *Given $\mathcal{B}$ for (R_Z, q) with $q \in Q_Z$, let $\mathcal{Z} = \{Z_1, \ldots, Z_k\}$, $Q = \{q_1, \ldots, q_h\}$, and $\tau = \frac{T}{N} \in \mathbb{R}^+$, we define:*

- *the associated* containment matrices $C^{\mathcal{Z}}_{\mathcal{B}}$ *and* $C^{Q}_{\mathcal{B}}$ *as the $(N+1) \times k$ and $(N+1) \times h$ matrices:*
$$C^{\mathcal{Z}}_{\mathcal{B}}(a,b) = \begin{cases} 1 & \textit{if } \bigcup_{a\tau \in J_i} Y_i \subseteq R_{Z_b} \wedge \bigwedge_{a\tau \in J_i} p_i \in Q_{Z_b} \\ 0 & \textit{otherwise} \end{cases} ; \quad C^{Q}_{\mathcal{B}}(a,b) = \begin{cases} 1 & \textit{if } \forall i \mid a\tau \in J_i, q_b = p_i \\ 0 & \textit{otherwise} \end{cases} ;$$
- *the associated* overapproximation matrices $O^{\mathcal{Z}}_{\mathcal{B}}$ *and* $O^{Q}_{\mathcal{B}}$ *as the $(N+1) \times k$ and $(N+1) \times h$ matrices:*
$$O^{\mathcal{Z}}_{\mathcal{B}}(a,b) = \begin{cases} 1 & \textit{if } R_{Z_b} \cap \bigcup_{a\tau \in J_i \wedge p_i \in Q_{Z_b}} Y_i \neq \emptyset \\ 0 & \textit{otherwise} \end{cases} ; \quad O^{Q}_{\mathcal{B}}(a,b) = \begin{cases} 1 & \textit{if } \exists i \mid a\tau \in J_i, q_b = p_i \\ 0 & \textit{otherwise} \end{cases} .$$

Informally, the a-rows of such matrices gives us information on where a solution that starts in (R_Z, q) can be at time $t = a\tau$ with respect to the local regions $\mathcal{Z}$: matrix C tracks containments and matrix O tracks intersections. We now transform this information in HLTL constraints that allow us to refine the behavior of the FTS.

Definition 13. *Given reachability matrices $C^{\mathcal{Z}}_{\mathcal{B}}, C^{Q}_{\mathcal{B}}, O^{\mathcal{Z}}_{\mathcal{B}}, O^{Q}_{\mathcal{B}}$, the associated a-*constraint *for $a = 0, \ldots, N$ is defined as:*
$\gamma_a = \bigwedge_{b \mid C^{\mathcal{Z}}_{\mathcal{B}}(a,b)=1} Z_b \wedge \bigwedge_{b \mid C^{Q}_{\mathcal{B}}(a,b)=1} q_b \wedge \bigwedge_{b \mid O^{\mathcal{Z}}_{\mathcal{B}}(a,b)=0} \neg(Z)_b \wedge \bigwedge_{b \mid O^{Q}_{\mathcal{B}}(a,b)=0} \neg q_b$.

Theorem 4. *Let $\mathcal{B}$ be a RSO for the system H for (R_Z, q), and let γ_a be the formulae built as in Definition 13 using the associated reachability matrices $C^{\mathcal{Z}}_{\mathcal{B}}, C^{Q}_{\mathcal{B}}, O^{\mathcal{Z}}_{\mathcal{B}}, O^{Q}_{\mathcal{B}}$. The system H satisfies the HLTL formula*

$$H \models G((Z \wedge q) \rightarrow (\gamma_0 \wedge (\gamma_0 U(\gamma_1 \wedge (\gamma_1 U(\gamma_2 \wedge \ldots \gamma_{N-2} U(\gamma_{N-1} \wedge (\gamma_{N-1} U \gamma_N))) \ldots)$$

Remark. It is often the case that several consecutive lines of $C^{\mathcal{Z}}_{\mathcal{B}}, C^{Q}_{\mathcal{B}}, O^{\mathcal{Z}}_{\mathcal{B}}$, and $O^{Q}_{\mathcal{B}}$ are equal. One can get rid of these multiple occurrences without losing expressive power of the formula presented in Theorem 4.

For each couple $Z \in \mathcal{Z}, q \in Q_Z$, we compute the RSO for $(R_Z, q)_{q \in Q_Z}$ and the associated reachability matrices. The constraint $\gamma_{Z,q}$ is the one that appears in Theorem 4.

Running example In Figure 3 we show two out of the 18 successfully computed RSOs for the running example. From the picture, it is clear that every trajectory that starts in R_{Z_2} reaches R_{Z_3} without intersecting B. We refrain from writing the full formula obtained applying Theorem 4; however, from it one can deduce: $H \models (Z_2 \wedge q_2) \to ((\neg B) U (Z_3 \wedge q_2))$, and similarly for Z_4, q_3 and Z_1, q_3. Notice that RSO computation may fail for certain local region (*e.g.* unbounded ones); in these cases $\gamma_{Z,q}$ is set to $\top$.

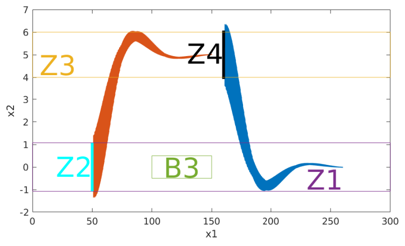

Fig. 3. Two RSO for the running example, starting from (Z_2, q_2) and (Z_4, q_3). In the plot, also projections on (x_1, x_2) for Z_1, Z_3 and part of B are depicted.

Remark 1. Certain classes of hybrid systems are stable iff there exist a piecewise quadratic Lyapunov function [DBPL00]. Since the verification of any LTL property can be reduced to the verification of a property of the form $F(G(p))$ on the hybrid system (in product with a discrete automaton), there is in principle no limitation on the form of the LTL properties that can be proven using this method. However, the problem is in general undecidable.

3.4 Discrete Abstraction

In this section, we define an FTS D_H that abstracts the behavior of the hybrid system H, given a set of local regions $\mathcal{Z}$. Informally, we approximate the predicate abstraction of H with respect to $\mathcal{Z}$, by computing the precise abstraction of the discrete structure of H and adding some simple constraints that capture the consistency and adjacency of the regions with respect to the locations of H. Additional constraints derived from the certificate functions and RSO (computed in Section 3.3) are added later on top of D_H.

Definition 14. *Given a HA $H = (X, Q, q_{init}, E, init, flow, inv, guard, jump)$, the discrete abstraction D_H of H w.r.t. $\mathcal{Z}$ is an FTS $\langle V, Init, Inv, Trans\rangle$ where:*

- *V is the set of Boolean variables $\{v_Z\}_{Z=(R_Z,Q_Z)\in\mathcal{Z}} \cup \{v_q\}_{q\in Q}$;*
- *$Init(V) = v_{(init,\{q_{init}\})}$;*
- *$Inv(V)$ is the conjunction of the following: 0) $\bigwedge_{Z\in\mathcal{Z}}\Big(v_Z \to \bigvee_{q\in Q_Z} v_q\Big)$;*
  1) $count(\{v_q\}_{q\in Q}) = 1$;
  2) $\bigwedge_{q\in Q}\Big(v_q \to \Big(\exists x. \bigwedge_{q\in Q_Z}(v_Z \leftrightarrow R_Z(x)) \wedge inv(q)(x)\Big)\Big)$;

- *$Trans(V, V')$ is the conjunction of the following:*
  3) $\bigwedge_{(q,q')\in E}((v_q \wedge v'_{q'}) \rightarrow \exists x, x' \,.\, inv(q)(x) \wedge inv(q')(x') \wedge jump(e)(x,x') \wedge guard(e)(x) \wedge \bigwedge_{q\in Q_Z}(R_Z(x) \leftrightarrow v_Z) \wedge \bigwedge_{q'\in Q_Z}(R_Z(x') \leftrightarrow v'_Z))$;
  4) $\left(\bigwedge_{q\in Q} v_q = v'_q\right) \rightarrow \bigwedge_{q\in Q} \wedge \bigwedge_{q\in Q_{Z_1}, q\in Q_{Z_2}}((v_q \wedge v_{Z_1} \wedge v'_{Z_2}) \rightarrow (\exists x \,.\, inv(q)(x) \wedge ((\overline{Z}_1(x) \wedge Z_2(x)) \vee (Z_1(x) \wedge \overline{Z}_2(x)))))$;

where:

- *the formula 1) is the Boolean encoding of the constraints that force one and only one variable in $\{v_q\}_{q\in Q}$ to be true;*
- *the quantifiers in formulas 2), 3), and 4) are removed with standard ALLSMT procedure [LNO06], optimized using static learning [Seb07].*

Given an HLTL formula γ, we denote by $\widehat{\gamma}$ the LTL formula obtained from γ by substituting each occurrence of q with v_q and each occurrence of Z with v_Z. This FTS is an abstraction of the hybrid automaton H in the following sense:

Theorem 5. *Let ψ be a HLTL property over $\mathcal{Z} \cup Q$. It holds:* $D_H \models \widehat{\psi} \implies H \models \psi$.

Details of the proof can be found in the Appendix; here, we provide some implications of the constraints in Definition 14 to clarify the intuition, relating to Figure 4:
- Constraint 0): if v_Z is true, a variable v_q with $q \in Q_Z$ must be true;
- Constraint 1): one location variable is true at each step;
- Constraint 2): empty combinations are not allowed, *e.g.* we cannot have $v_{loc_1} \wedge v_{Z_1} \wedge v_{Z_2}$;
- Constraint 3): the set of true variables before and after a transition where at least one location variable changes its value must be consistent with the jump and guard conditions, *e.g.* we do not admit $(v_{loc_1} \wedge v_{Z_2}) \wedge (v_{loc_2} \wedge v_{Z_2})'$, but we admit $(v_{loc_1} \wedge v_{Z_1}) \wedge (v_{loc_2} \wedge v_{Z_2})'$;
- Constraint 4): we do not admit a transition within a location among two regions that are not adjacent, *e.g.* we do not admit $(v_{loc_1} \wedge v_{Z_1}) \wedge (v_{loc_1} \wedge v_{Z_2})'$, but we admit $(v_{loc_1} \wedge v_{Z_1}) \wedge (v_{loc_1} \wedge v_{Z_3})'$;

The following is a consequence of Theorems 1 - ... - 5:

Corollary 1. *If $D_H \models (\bigwedge_{f\in\mathcal{F}} \widehat{\gamma_f} \wedge \bigwedge_{Z\in\mathcal{Z}, q\in Q_Z} \widehat{\gamma_{Z,q}}) \rightarrow \widehat{\phi}$, then $H \models \phi$.*

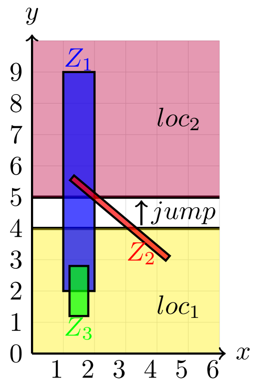

Fig. 4. A configuration of local regions in a hybrid system, used to explain the constraints in Definition 14. Invariant sets of locations are higlighted, and the jump condition is $x' = x \wedge y' = y + 1$.

3.5 Computation of Certificate Functions and Regions

We now detail the first three steps of the algorithm presented in Section 3.1.
Lyapunov-like and Descent functions We briefly describe the synthesis of a Lyapunov-like function for the system H, and refer to Appendix E (in particular to Algorithm 2) for further details.

Given a subset S of variables, we define $flow(q)|_S$ the coordinates of $flow(q)$ corresponding to the variables in S. Given a location q and a subset S of variables such that the formulae $flow(q)|_S$ contain only variables in S, we look for a classic Lyapunov function for $flow(q)$ restricted to the variables S.

Let z_e be an equilibrium point for $flow(q)|_S$. The candidate Lyapunov function $V(z)$ is constructed as a polynomial of a specified degree d by requiring: $V(z) > 0 \ \wedge \ \dot{V}(z) < 0 \quad \text{for all } z_e \neq z \in inv(q)$, These conditions can be given to an LMI or an SOS solver. Once the function V is synthesized, we validate it through the use of an SMT solver. If either the synthesis or the validation fails, we try to synthesize a Lyapunov function by finding a solution to a symbolic equation. If also this one fails, we move on with a different subset of variables or a different location. Notice that the requirements on V imply that the extension of V to the set of all variables is a Lyapunov-like function as in Definition 7 for the system H with $L_Q = \{q\}$.

If the Lyapunov-like function is found and validated, we try to expand L_Q by picking another location q_{new} and checking if the dynamics restricted to S is still independent from the other variables, verifying whether conditions on V holds in q_{new}, and whether it does not increase on jumps from q to q_{new} (or vice versa). If this is the case, we can add q_{new} to L_Q and repeat the process for other locations, until we cannot add any other location.

The synthesis of a descent function is similar to the synthesis of a Lyapunov-like function, by modifying the requirements according to the definition.
Region synthesis and Barrier Functions Starting with the Lyapunov-like functions that we have found, we create a list of local regions to be considered in the discrete abstraction. We here outline the method, and refer to the complete Algorithm 3 in Appendix for the details. For each valid Lyapunov-like function (L_f, L_Q) synthesized, we create a list of *critical values* $CV_{(L_f, L_Q)}$. These are obtained by computing maxima and minima of L_f on the local regions $\mathcal{Z}_{in}$ given in input that share a location with L_Q, and adding the midpoints of two consecutive critical values. The local regions to add to the abstraction are the sublevels of the Lyapunov-like function given by these values: $\big\{(L_f \leq a, L_Q) \mid a \in CV_{(L_f, L_Q)}\big\}$. For each such local region, we add the *associated barrier function* $(L_f - a,\ L_Q)$ that is a valid barrier certificate and a witness that once a sublevel is reached, the system will not leave it. As a final step, for each local region $(L_f \leq a, L_Q)$ and each $e = (q_1, q_2) \in E$ such that $q_1 \in L_Q$, we compute the image of $(L_f \leq a) \cap guard(e)$ through $jump(e)$, and add it to the list of local regions the ones that are non-empty.

4 Experimental evaluation

4.1 Overview

In this section, we present six experiments designed to evaluate the effectiveness and robustness of the proposed algorithm in proving liveness properties of hybrid systems. We apply out method to: 1) Linear Overtake (the running example

Benchmark	**dim**	$\|Q\|$	**flow**	$\|\mathcal{Z}_\phi\|$	$\|\mathcal{Z}_{add}\|$
Linear Overtake	4	3	Linear	3	0
Multiple choice example	2	$2+2n$	Linear	1	0
Navigation Benchmark	4	25	Linear	1	14
Non-linear circular	2	1	Nonlinear	2	11
Multilayer control	3	n	Lin - Nonlin	2	0

Table 2. Characteristics of the six benchmarks. When $|\mathcal{Z}_{add}| = 0$, we did not need any additional handcrafted region to prove the properties.

presented in Section 3.2); 2) Multiple choice: a scalable benchmark; 3) Navigation Benchmark: an instance of the benchmark from [FI04]; 4) Non-linear circular dynamics: a system with one location and non-linear dynamics converging to the unit circle; 5-6) Multilayer Control (linear and nonlinear): a hybrid system where the location decides the point to which the system converges and the dynamics can be linear or non-linear. Table 2 reports for each benchmark the number of continuous variables, the number of locations, the type of flow conditions, and the cardinality $|\mathcal{Z}_{add}|$. For each benchmark, we run the algorithm in four configurations: with or without adding images through $jump$ functions to $\mathcal{Z}_{Lyap}$ during Step 2, and using or not using stabilizing constraints [CS12] in the model checking. Adding images through $jump$ functions results in an increased time execution for the algorithm, but it is sometimes necessary to be able to prove the properties.
How to read the tables The complete logs from the benchmarks are available in the artifact evaluation [BT]. We report in the tables the most interesting entries. They are the time used for, respectively: T^{ic3-e}_{TOT}: total using IC3 without stabilizing constraints; T^{ic3}_{TOT}: total using with IC3 with stabilizing constraints; $T^{vb}_{TOT} := \min\{T^{ic3-e}_{TOT}; T^{ic3}_{TOT}\}$; T_{Lyap}: Lyapunov-like function; T_{ASJump}: AllSMT with discrete jumps - formula 3) in Definition 14; T_{ASReg}: AllSMT within a location - formula 2) in Definition 14; T_{AdReg}: adjacency constraints - formula 4) in Definition 14; T_{RSO}: the RSOs; T_{RSOMat}: the RSO matrices; T_{ic3} - T_{ic3-e}: to solve the model checking problem with k-liveness and IC3 using (resp. not using) stabilizing constraints [CS12]; $T_{vb} := \min\{T_{ic3}; T_{ic3-e}\}$. Furthermore, we provide the average of: md_{RSO}: temporal depth of LTL properties coming from the RSO; ms_{RSO}: size of LTL properties coming from the RSO; ms_{bar}: size of LTL properties coming from the barriers. We also show: N_{LTL}: the number of LTL constraints synthesized by the algorithm; $|\mathcal{Z}|$: the number of regions in the set $\mathcal{Z}$. Notice that the value of the temporal depth and size that are not reported may vary only up to ± 1 because of the way such LTL constraints are built.

Algorithm `LTLConstraints4HS` is implemented using an extension of pySMT [GM15] for VMT format [VMT11] called pyVMT [pyV22] for the model construction, z3 [dMB08] and cvc5 [BBB$^+$22] to check the validity of the polynomial conditions, nuXmv [CCD$^+$14] for model checking and CORA [Alt15] inside MATLAB [Inc22] for RSO with $T = 10$ and $\tau = 0.1$ or 0.01 depending on the complexity of the system, and for computation of reachability matrices. We ran

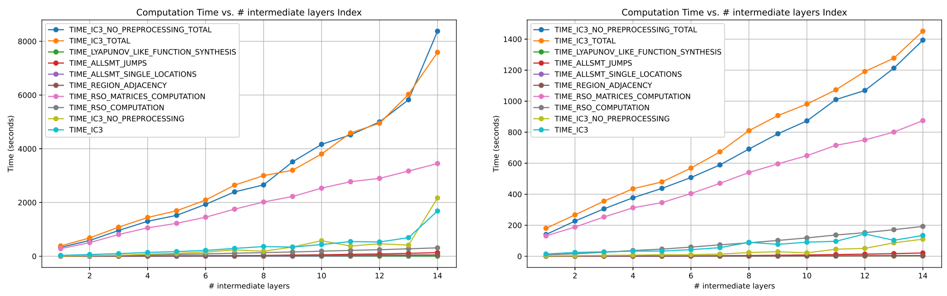

Fig. 5. Computation times for multiple choice benchmark, with (left) and without (right) images through jump functions.

experiments on a machine with 32 GB of RAM and a 12-core, 3.5 GHz processor. You can find additional information in the Appendix C and at [BT].

4.2 Benchmarks

Linear Overtake We already presented the benchmark in Section 3.2. We apply our algorithm to the properties: the car gets infinitely often to the initial location, *i.e.* $p_1 = G(F(q_1))$, then $p_2 = \phi$ as defined in Section 3.2, and that the bad states are never reached, *i.e.* $p_3 = G(\neg B)$. Results are shown in Table 3. We were not able to prove p_2 and p_3 without considering images through jumps, and using stabilizing constraints in the model checking resulted in a timeout.

T^{vb}_{TOT}	T_{Lyap}	T_{ASJump}	T_{ASReg}	T_{AdReg}	T_{RSOMat}	T_{RSO}	$T_{vb}p_1$	$T_{vb}p_2$	$T_{vb}p_3$	md_{RSO}	ms_{RSO}	ms_{bar}	N_{LTL}	$\lvert\mathcal{Z}\rvert$
359.88	0.45	4.11	1.37	32.85	93.63	4.07	68.94	49.70	6.72	15.50	532.17	68.33	153	40

Table 3. Data for linear overtake benchmark with images through jump functions.

Multiple choice This 2-dimensional linear hybrid system MC_n schematize a 2-dimensional system that moves with $\dot{x_1} = 10$ and that can have a discrete jump after traveling a fixed distance in the x_1-coordinate: to move close to $x_2 = 5$ or close to $x_2 = 0$. The parameter n decides how many times this choice can be made before arriving in a final location with globally asymptotically stable dynamics. Precise definitions and complete results are given in Appendix C.2, together with some figures to visualize the system. The initial local region is $Z_0 = ([-1, 1] \times [-1, 1], Loc_1)$. We want to prove that we eventually end up in the local region $Z_{target} = ([5(n+2)-1, 5(n+2)+1] \times [-1, 1], Loc_{2n+2})$, therefore we define property $p_1 = F(G(Z_{target}))$. Results are shown in Figure 5. For $n = 14$ with images through jumps, we have $N_{LTL} = 1223$ and $|\mathcal{Z}| = 169$.

Navigation Benchmark This example is taken from [FI04, Section 2.1] and its simplification is used as a benchmark for the ARCH-competition [BKRS24]. We briefly describe the system in Appendix C.3, and refer to the original paper for the full definition of the hybrid automaton and the *map* of a specific instance. It consists in a 4-dimensional hybrid system with linear intricate dynamics, where proving the existence of recurrent trajectories is not trivial. This demonstrates how our algorithm can verify complex properties when appropriate regions are carefully selected. We consider the system corresponding to the instance with the map: $[4,5,6,6,6;4,3,2,0,6;4,3,2,0,6;4,3,2,0,6;2,2,1,0,7]$. Some simulations of the system are presented in Appendix C.3. Based on these, we defined the local region $Z_0 = ([3.49;3.55]\times[4.26;4.32]\times[-0.35;-0.29]\times[0.65;0.69],\{Loc_3\})$ as the initial local region, for which we want to prove the liveness property $p_1 := G(F(Z_0))$. In order to be able to do this, we needed the regions $\mathcal{Z}_{add}$ that can be found in Appendix C.3. Results are presented in Table 4. Considering images through jumps or stabilizing constraints resulted in timeout.

T^{vb}_{TOT}	T_{Lyap}	T_{ASJump}	T_{ASReg}	T_{AdReg}	T_{RSOMat}	T_{RSO}	$T_{vb}p_1$	md_{RSO}	ms_{RSO}	ms_{bar}	N_{LTL}	$\|\mathcal{Z}\|$
15633.64	6.54	79.88	11.00	33.75	177.83	20.98	14090.35	35.67	2273.33	148.64	222	37

Table 4. Data for navigation benchmark, without images through jump functions.

Non-linear circular The fourth experiment explores a non-linear system with only one location aiming to prove the existence of a loop that avoids a specified bad region. The dynamics is given by:
$\dot{x_1} = -x_1\left(x_1^2+x_2^2-1\right) - x_2;\quad \dot{x_2} = -x_2\left(x_1^2+x_2^2-1\right) + x_1.$
Notice that regions and local regions are equivalent when we have only one location. The starting region is $R_{0,\alpha} = [1-\alpha;1+\alpha]\times[-\alpha;\alpha]$ for the values $\alpha = 0.05, 0.08, 0.11$. We also define a bad region $B = [0.3;0.5]\times[-0.7;-0.5]$, that we want to prove that we avoid. The other regions in $\mathcal{Z}_{add}$ can be found in the Appendix C.4. We prove the properties $p_1 := G(F(R_{0,\alpha}))$ and $p_2 := G(\neg B)$. Results are shown in Table 5. Notice that in this case we have no jumps, so there is no difference in considering images through them or not. Performances of $ic3$ and $ic3-e$ are similar in this case.

α	T^{vb}_{TOT}	T_{Lyap}	T_{ASReg}	T_{AdReg}	T_{RSOMat}	T_{RSO}	$T_{vb}p_1$	$T_{vb}p_2$	md_{RSO}	ms_{RSO}	N_{LTL}	$\|\mathcal{Z}\|$
0.05	182.03	0.15	0.04	3.54	107.50	85.64	15.59	0.70	12.00	255.67	6	13
0.08	115.70	0.14	0.04	3.56	31.65	66.53	21.92	0.49	12.00	263.00	2	13
0.11	103.01	0.14	0.04	3.44	32.38	49.42	21.10	0.41	12.00	255.00	2	13

Table 5. Computation times for nonlinear circular benchmark.

Multilayer control The last benchmark consists in a hybrid systems on 3 variables (x,y,t) with $n \geq 4$ locations. Let $\{o_i\}_{i=1,\dots,n}$ be the vertices of the regular n-gon centered in the origin ordered counterclockwise with $o_1 = (1,0)$. The flow $flow(q_i)$ converges to o_i. We consider two configurations: one where the dynamics is linear ($\dot{x} = -x + x_{o_i}, \dot{y} = -y + y_{o_i}, \dot{t} = 1$) and one where the dynamics is non-linear ($\dot{x} = (-x + x_{o_i})^3, \dot{y} = (-y^3 + y_{o_i})^3, \dot{t} = 1$). A switch may happen between any pair of locations when $t \geq 10$, and the reset function is $x' = x, y' = y, t' = 0$. The starting local region is $Z_0 = ([0.9,1.1]\times[-0.1,0.1]\times\mathbb{R}, q_1)$. Let $B = ([-0.1,0.1]\times[-0.1,0.1], Q)$. We check two properties: that the bad region is avoided if switches may only occur among consecutive locations, *i.e.*
$p_1 := \Big(G\Big(\bigwedge_{i=1}^{n}\Big(q_i \rightarrow \Big(\Big(\bigwedge_{j=1,\dots,n;j\neq i-1,i,i+1}\neg q_j\Big)W(q_{i-1}\vee q_{i+1})\Big)\Big)\Big)\Big) \rightarrow G(\neg B)$,
where $q_{-1} := q_n$ and $q_{n+1} := q_1$; and that if a solution eventually remains in q_1, it will get and remain in Z_0, *i.e.* $p_2 := G(G(q_1) \rightarrow F(G(Z_0)))$.

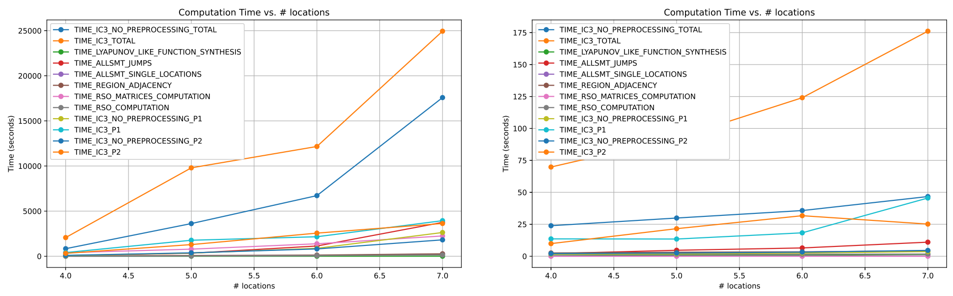

Fig. 6. Computation times for multilayer linear control benchmark, with (left) and without (right) images through jump functions.

As far as the linear control is concerned, we do not need images through jumps to prove p_2, while we are not able to prove p_1 without them. When considering the nonlinear control, we are not able to prove p_1 even with images through jumps. The difference between these two cases is given by the RSO, that is much more complex in the nonlinear case and struggles to prove that trajectories after the switch does not intersect B. Results are shown in Figure 6 (linear case) and in Appendix C.4, Figures 15-16 (nonlinear case).

5 Conclusions and Future Works

In this paper, we addressed the problem of verifying liveness properties, expressed in LTL, on hybrid systems, with linear or non-linear dynamics. The method leverages Lyapunov-like functions and reachable set overapproximation to synthesize regions and LTL properties to be used for the discrete-time abstraction. We then use standard LTL model checking techniques to prove the property on the abstraction. We implemented the approach on top of the CORA and nuXmv tools, showing how it can scale in the size of the discrete structure and how it can prove properties over linear and non-linear complex dynamics. The work paves the way to different future directions. The automation for the set of regions, the set of functions, and the parameters for the reachable set overapproximation can be improved; it is an open challenge to fully automate and guide their generation for finding the inputs sufficient to prove the property. This would be very important specifically for barrier functions, whose computation without initial/unsafe region can be challenging. Several tools and ideas developed along these lines could be integrated into the workflow [APA20,BT24]. Complexity could be addressed in many points: semi-algebraic reasoning can be very expensive, and LTL-to-automata translation can be probably optimized, even though the symbolic approach of nuXmv is already very efficient. We will consider the generalization of the properties with metric operators as in MTL or STL, for example leveraging the exponential decay of Lyapunov functions.

References

[ACH$^+$95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The Algorithmic Analysis of Hybrid Systems. *Theor. Comput. Sci.*, 138(1):3–34, 1995.

[ADI06] Rajeev Alur, Thao Dang, and Franjo Ivančić. Counterexample-guided predicate abstraction of hybrid systems. *Theoretical Computer Science*, 354(2):250–271, 2006. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003).

[AFG21] M. Althoff, G. Frehse, and A. Girard. Set propagation techniques for reachability analysis. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1):369–395, 2021.

[AGRS25] Alessandro Abate, Mirco Giacobbe, Diptarko Roy, and Yannik Schnitzer. *Model Checking and Strategy Synthesis with Abstractions and Certificates*, pages 360–391. Springer Nature Switzerland, Cham, 2025.

[AHLP00] R. Alur, T.A. Henzinger, G. Lafferriere, and G.J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.

[Alt10] Matthias Althoff. *Reachability Analysis and its Application to the Safety Assessment of Autonomous Cars.* PhD thesis, 07 2010.

[Alt15] Matthias Althoff. An introduction to CORA 2015. In *Proc. of the 1st and 2nd Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151. EasyChair, December 2015.

[AMTZ21] Mahathi Anand, Vishnu Murali, Ashutosh Trivedi, and Majid Zamani. Safety verification of dynamical systems via k-inductive barrier certificates. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 1314–1320, 2021.

[APA20] Daniele Ahmed, Andrea Peruffo, and Alessandro Abate. Automated and Sound Synthesis of Lyapunov Functions with SMT Solvers. In *TACAS (1)*, volume 12078 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2020.

[Bak18] Stanley Bak. t-barrier certificates: A continuous analogy to k-induction. *IFAC-PapersOnLine*, 51(16):145–150, 2018. 6th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2018.

[BBB$^+$22] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022.

[BFF$^+$19] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin, and Christian Schilling. Juliareach: a toolbox for set-based reachability. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, page 39–44, New York, NY, USA, 2019. Association for Computing Machinery.

[BKRS24] Lei Bu, Atanu Kundu, Rajarshi Ray, and Yuhui Shi. Arch-comp24 category report: Hybrid systems with piecewise constant dynamics and bounded model checking. In Goran Frehse and Matthias Althoff, editors, *Proceedings of the 11th Int. Workshop on Applied Verification for Continuous and Hybrid Systems*, volume 103 of *EPiC Series in Computing*, pages 1–14. EasyChair, 2024.

[Bra94] M.S. Branicky. Stability of switched and hybrid systems. In *Proceedings of 1994 33rd IEEE Conference on Decision and Control*, volume 4, pages 3498–3503 vol.4, 1994.

[BT] Ludovico Battista and Stefano Tonetta. Artifact evaluation link. `https://zenodo.org/records/15846315`.

[BT24] Ludovico Battista and Stefano Tonetta. Formal verification of stability for parametric affine switched systems. *IFAC-PapersOnLine*, 58(11):37–42, 2024. 8th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2024.

[CBGV12] Pieter Collins, Davide Bresolin, Luca Geretti, and Tiziano Villa. Computing the evolution of hybrid systems using rigorous function calculus*. *IFAC Proceedings Volumes*, 45(9):284–290, 2012. 4th IFAC Conference on Analysis and Design of Hybrid Systems.

[CCD$^+$14] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In Armin Biere and Roderick Bloem, editors, *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 334–342. Springer, 2014.

[CGMT14] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Verifying LTL Properties of Hybrid Systems with K-Liveness. In *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 424–440. Springer, 2014.

[CN12] Rebekah Carter and Eva M. Navarro-López. Dynamically-driven timed automaton abstractions for proving liveness of continuous systems. In Marcin Jurdzinski and Dejan Nickovic, editors, *Formal Modeling and Analysis of Timed Systems - 10th International Conference, FORMATS 2012, London, UK, September 18-20, 2012. Proceedings*, volume 7595 of *Lecture Notes in Computer Science*, pages 59–74. Springer, 2012.

[CRT09] Alessandro Cimatti, Marco Roveri, and Stefano Tonetta. Requirements validation for hybrid systems. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, pages 188–203, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[CS12] Koen Claessen and Niklas Sörensson. A liveness checking algorithm that counts. In *FMCAD*, pages 52–59. IEEE, 2012.

[dAM95] Luca de Alfaro and Zohar Manna. Verification in continuous time by discrete reasoning. In V. S. Alagar and Maurice Nivat, editors, *Algebraic Methodology and Software Technology*, pages 292–306, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[DBPL00] R.A. Decarlo, M.S. Branicky, S. Pettersson, and B. Lennartson. Perspectives and results on the stability and stabilizability of hybrid systems. *Proceedings of the IEEE*, 88(7):1069–1082, 2000.

[dC16] Julien Alexandre dit Sandretto and Alexandre Chapoutot. Validated explicit and implicit Runge–Kutta methods. *Reliable Computing*, 22(1):79–103, Jul 2016.

[dMB08] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[FI04] Ansgar Fehnker and Franjo Ivančić. Benchmarks for hybrid systems verification. In Rajeev Alur and George J. Pappas, editors, *Hybrid Systems: Computation and Control*, pages 326–341, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[FMQ$^+$15] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In Amy P. Felty and Aart Middeldorp, editors, *CADE*, volume 9195 of *LNCS*, pages 527–538. Springer, 2015.

[GG08] Antoine Girard and Colas Le Guernic. Efficient reachability analysis for linear systems using support functions. *IFAC Proceedings Volumes*, 41(2):8966–8971, 2008. 17th IFAC World Congress.

[GM15] Marco Gario and Andrea Micheli. Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In *SMT Workshop 2015*, 2015.

[GST12] R. Goebel, R. G. Sanfelice, and A. R. Teel. *Hybrid Dynamical Systems: Modeling, Stability, and Robustness.* Princeton University Press, New Jersey, 2012.

[Hen96] T.A. Henzinger. The theory of hybrid automata. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292, 1996.

[HS20] Hyejin Han and Ricardo G. Sanfelice. Linear temporal logic for hybrid dynamical systems: Characterizations and sufficient conditions. *Nonlinear Analysis: Hybrid Systems*, 36:100865, 2020.

[Inc22] The MathWorks Inc. Matlab version: 9.13.0 (r2022b), 2022.

[Kel14] Christopher M. Kellett. A compendium of comparison function results. *Mathematics of control, signals, and systems*, 26(3):339–374, 2014.

[Kha02] Hassan K Khalil. *Nonlinear systems; 3rd ed.* Prentice-Hall, Upper Saddle River, NJ, 2002. The book can be consulted by contacting: PH-AID: Wallet, Lionel.

[LNO06] Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras. SMT Techniques for Fast Predicate Abstraction. In *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 424–437. Springer, 2006.

[NEDR$^+$23] Satya Prakash Nayak, Lucas N. Egidio, Matteo Della Rossa, Anne-Kathrin Schmuck, and Raphael M. Jungers. Context-triggered abstraction-based control design. *IEEE Open Journal of Control Systems*, 2:277–296, 2023.

[Oeh11] Jens Oehlerking. *Decomposition of stability proofs for hybrid systems.* PhD thesis, Universität Oldenburg, 2011.

[PA18] Christian Pek and Matthias Althoff. Efficient computation of invariably safe states for motion planning of self-driving vehicles. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3523–3530, 2018.

[Par00] Pablo Parrilo. Structured semidenite programs and semialgebraic geometry methods in robustness and optimization. *PhD thesis*, 08 2000.

[PKV13] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Falsification of LTL safety properties in hybrid systems. *Int. J. Softw. Tools Technol. Transf.*, 15(4):305–320, 2013.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

[pyV22] PyVmt: a python library to interact with transition systems. `https://github.com/pyvmt/pyvmt`, 2022. [Github repository].

[SÁW$^+$24] Stefan Schupp, Erika Ábrahám, Md Tawhid Bin Waez, Thomas Rambow, and Zeng Qiu. On the applicability of hybrid systems safety verification tools from the automotive perspective. *Int. J. Softw. Tools Technol. Transf.*, 26(1):49–78, 2024.

[Seb07] Roberto Sebastiani. Lazy Satisability Modulo Theories. *J. Satisf. Boolean Model. Comput.*, 3(3-4):141–224, 2007.

[VMT11] VMT-LIB: Verification Modulo Theories (Language, Benchmarks and Tools). `https://vmt-lib.fbk.eu/`, 2011.

[WS11] Rafael Wisniewski and Christoffer Sloth. Abstraction of dynamical systems by timed automata. *Modeling, Identification and Control (Online Edition)*, 32(2):79–90, 2011.

[WTL14] Tichakorn Wongpiromsarn, Ufuk Topcu, and Andrew G. Lamperski. Automata theory meets barrier certificates: Temporal logic verification of nonlinear systems. *IEEE Transactions on Automatic Control*, 61:3344–3355, 2014.

[ZZCL20] Hengjun Zhao, Xia Zeng, Taolue Chen, and Zhiming Liu. Synthesizing barrier certificates using neural networks. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, HSCC '20, New York, NY, USA, 2020. Association for Computing Machinery.

A Running example details

The precise formal model of the example of Sec. 3.2 is described below:

– the set Q is given by three locations $\{q_1, q_2, q_3\}$ that represent the moments before, during, and after the overtake, the initial location is q_1;
– $init := x_1 \leq -100 \wedge -0.1 \leq x_2 \leq 0.1 \wedge 9.9 \leq v_1 \leq 10.1 \wedge -0.1 < v_2 < 0.1$;
– the set of transitions is given by $E = \{(q_1, q_2), (q_2, q_3), (q_3, q_1)\}$;
– invariant sets are defined as: $inv(q_1) = \{x_1 \leq -50, v_1 > 0.1\}$; $inv(q_2) = \{-50 \leq x_1 \leq 60, v_1 > 0.1\}; inv(q_3) = \{60 \leq x_1 \leq 200, v_1 > 0.1\}$;
– flow is defined as:
$flow(q_3) = flow(q_1) = \dot{x_1} = v_1,\ \dot{x_2} = v_2,\ \dot{v_1} = -(v_1 - 10),\ \dot{v_2} = -x_2 - v_2$;
$flow(q_2) = \dot{x_1} = v_1,\ \dot{x_2} = v_2,\ \dot{v_1} = -(v_1 - 10),\ \dot{v_2} = -(x_2 - 5) - v_2$;
– guards are defined as: $guard((q_1, q_2)) = (x_1 = -50)$;
$guard((q_2, q_3)) = (x_1 = 60); guard((q_3, q_1)) = (x_1 = 200)$;
– jumps are defined as:
$jump((q_1, q_2)) = jump((q_2, q_3)) = (x_1 = x_1' \wedge x_2 = x_2' \wedge v_1 = v_1' \wedge v_2 = v_2')$;
$jump((q_3, q_1)) = (x_1' \leq -50 \wedge x_2 = x_2' \wedge v_1 = v_1' \wedge v_2 = v_2')$;

B Proofs

Here we list the proofs of the statements given in the paper.

Proof (Theorem 1).
We start by proving the following lemma:

Lemma 1. *Let $L = (L_f, L_Q)$ be a Lyapunov-like function for H. Let $\sigma = \langle f_i, I_i, q_i \rangle$ be a run of H such that, for all $i \geq 0$, $loc(\sigma_i) \in L_Q$. Let (x_i) be any sequence of points such that $(x_i) \in \operatorname{Im} f_i$. It holds: $\lim_{i\to\infty} L_f((x_i, q_i)) = 0$. In particular, if R and q are such that $\inf_{(R\cap inv(q))\times\{q\}} L_f > 0$, any solution will eventually leave R or q.*

Proof. Conditions $\mathcal{L}_{flow(q)} L_f(x,q) \leq 0$ and $L_f((x', q')) \leq L_f((x,q))$ imply that the value of L_f is decreasing along trajectories. Since it is bounded below by zero, there exists some limit $\lim_{i\to\infty} L_f(x_i, q_i) = c \geq 0$. Assume, for the sake of contradiction, that $L_f((x_i, q_i))$ converges to a positive constant $c > 0$ as $i \to \infty$. Since α is a $\mathcal{K}$-class function, $\alpha(c) > 0$. The condition $\mathcal{L}_{flow(q)} L_f(x,q) \leq -\alpha(L_f(x,q))$ implies that for sufficiently large i, the derivative of $L_f(x,q)$ is strictly negative on non-degenerate intervals, i.e., $\frac{d}{dt} L_f f_i(t) < -\alpha(c) < 0$. Since we assume that the union of I_i covers $\mathbb{R}^{\geq 0}$, their lengths sum to infinity. This means that $L_f(x,q)$ decreases indefinitely, which contradicts the assumption that it converges to a positive constant c. Therefore $L_f((x_i, q_i)) \xrightarrow{i\to\infty} 0$. □

Proof of Theorem 1 is a direct consequence of Lemma 1: if a trajectory always stays in a location in L_Q (*i.e.*, $G(\bigvee_{p\in L_Q} p)$), then the value of the Lyapunov function will approach 0. If it is bounded by a positive constant on $(R_Z \cap inv(q)) \times \{q\}$, the solution will leave such set. □

Proof (Theorem 2). The proof is analogue to the one of Theorem 1, given the following lemma:

Lemma 2. *Let $D = (D_f, D_Q)$ be a descent function for H. Let σ be an execution of H such that $loc(\sigma_i) \in D_Q$ for all i. Let (x_i) be any sequence of points such that $(x_i) \in \operatorname{Im} f_i$. It holds: $\lim_{i\to\infty} D_f((x_i, q_i)) = -\infty$. In particular, if R and q are such that $\inf_{(R\cap inv(q))\times\{q\}} D_f > -\infty$, any solution will eventually leave R or q.*

Proof (Theorem 3). The proof is analogue to the one of Theorem 1, given the following lemma:

Lemma 3. *Let $B = (B_f, B_Q)$ be a barrier function for H. Let σ be an execution of H such that $loc(\sigma_i) \in B_Q$ for all i. Let (x_i) be any sequence of points such that $(x_i) \in \operatorname{Im} f_i$. Assume that exists i such that $B_f(x_i, loc(\sigma_i)) \leq 0$. Then: $\forall j \geq i\ B_f(x_j, loc(\sigma_j)) \leq 0$. In particular, if $R_1, R_2 \in Expr(X)$ and $q_1, q_2 \in B_Q$ are such that $R_1 \wedge inv(q_1) \to B_f(q_1) > 0$, and $R_2 \wedge inv(q_2) \to B_f(q_2) \leq 0$, any trajectory that starts in R_2 in location q_2 will not get to R_1 in location q_1.*

Proof (Theorem 4).

We prove the property for local regions of the form $(R_Z, Q_Z = Q)$. The general version can be obtained by using the fact that $C^{\mathcal{Z}}_{\mathcal{B}}(a,b) = 1$ implies that any run that starts in (R_Z, q) is in a location inside Q_{Z_b} at time $a\tau$, and that if every run at time $a\tau$ is in a location outside Q_{Z_b}, then $O^{\mathcal{Z}}_{\mathcal{B}}(a,b) = 0$.

Let $R := R_Z$. We prove the properties on runs that starts in $R \wedge q$. This implies the result because the system that we consider is autonomous.

We start by proving two lemmas.

Lemma 4. *Let $\sigma = \langle f_i, I_i, q_i \rangle$ be a run of H that is ground for $\mathcal{Z}$ such that $f_0(0) \in R, q_0 = q$. Let $k\tau \in I_i$. It holds:*

$$\sigma, i \models \gamma_k.$$

Proof. Since σ is ground for $\mathcal{Z}$, it is enough to check that

$$f_i(k\tau) \in \bigcap_{b \mid C^{\mathcal{Z}}_{\mathcal{B}}(a,b)=1} R_b \cap \bigcap_{b \mid O^{\mathcal{Z}}_{\mathcal{B}}(a,b)=0} \neg R_b \tag{1}$$

and that

$$q_i \in \{q_b\}_{b \mid C^{Q}_{\mathcal{B}}(a,b)=1} \wedge q_i \notin \{q_b\}_{b \mid O^{Q}_{\mathcal{B}}(a,b)=0}. \tag{2}$$

By the definition of RSO, to prove Condition (1) it is enough to show that

$$\bigcup_{i \;\mid\; k\tau \in J_i} Y_i \subseteq \bigcap_{b \mid C^{\mathcal{Z}}_{\mathcal{B}}(a,b)=1} R_b \cap \bigcap_{b \mid O^{\mathcal{Z}}_{\mathcal{B}}(a,b)=0} \neg R_b.$$

Both this and Condition (2) are immediate consequences of the definition of reachability matrices. □

Lemma 5. *Let $\sigma = \langle f_i, I_i, q_i \rangle$ be a run of H that is ground for $\mathcal{Z}$ such that $f_0(0) \in R, q_0 = q$. Let $t \in ((k-1)\tau, ((k+1)\tau)) \cap I_i$. It holds:*

$$\sigma, i \models \gamma_k.$$

Proof. As in the previous case, we can check the containment only for $f_i(k\tau)$.

The condition on the locations is again a consequence of the definition of RSO and reachability matrices.

By Definition 11, we have the containment $\{i \mid t \in J_i\} \subseteq \{i \mid k\tau \in J_i\}$. This means that the set used to compute reachability matrices for $k\tau$

$$\bigcup_{i \;\mid\; k\tau \in J_i} Y_i$$

contains the set

$$\bigcup_{i \;\mid\; t \in J_i} Y_i.$$

We can then conclude using the argument in the previous proof. □

We are left to prove the main result. Let γ be the property of interest. Consider a run σ that is ground for $\mathcal{Z}$. Consider a refinement $\rho = \langle g_i, J_i, p_i \rangle$ such that every J_i is contained in one of the following intervals:

$$\{0\},\ (0,\tau),\ \{\tau\},\ (\tau, 2\tau),\ \{2\tau\},\ (2\tau, 3\tau), \ldots$$

This is can be achieved using intersections. Since ρ is a refinement of σ, $\sigma \models \gamma$ if and only if $\rho \models \gamma$ (see [CRT09, Theorem 2]).

Any sequence of times $t_i \in J_i$ will be of the form:

$$\begin{aligned} t_0, \ldots, t_{k_0} &\in \{0\}, \\ t_{k_0+1}, \ldots, t_{k_1} &\in (0, \tau), \\ t_{k_1+1}, \ldots, t_{k_2} &\in \{\tau\}, \\ t_{k_2+1}, \ldots, t_{k_3} &\in (\tau, 2\tau), \\ t_{k_3+1}, \ldots, t_{k_4} &\in \{2\tau\}, \\ \vdots \qquad & \qquad \vdots \end{aligned}$$

By the previous lemmas, we know that

$$\begin{gathered} \rho, 0 \models \gamma_0 \wedge \ldots \wedge \rho, k_0 \models \gamma_0, \\ \rho, k_0+1 \models \gamma_0 \wedge \gamma_1 \wedge \ldots \wedge \rho, k_1 \models \gamma_0 \wedge \gamma_1, \\ \rho, k_1+1 \models \gamma_1 \wedge \ldots \wedge \rho, k_2 \models \gamma_1, \\ \rho, k_2+1 \models \gamma_1 \wedge \gamma_2 \wedge \ldots \wedge \rho, k_1 \models \gamma_1 \wedge \gamma_2, \\ \vdots \end{gathered}$$

And this proves that $\rho \models \gamma$. □

Remark 2. We also underline that this proposition admits a straightforward generalization to the case where endpoints are not contained in $\{k\tau\}_{k=0,\ldots,N}$ in Definition 11; this can be obtained by computing the intersections and the containments with regions in $\mathcal{Z}$ in the set of all endpoints of intervals in $\mathcal{B}$. We considered this easier case both to reflect the implementation and for clarity of exposition.

Proof (Theorem 5). We get the result by proving that if there is an initial run $\sigma = \langle f_i, I_i, q_i \rangle$ ground for $\{R_Z \mid Z \in \mathcal{Z}\}$ such that $\sigma \models \neg\phi$, then there is a run $\widehat{\sigma}$ of D_H such that $\widehat{\sigma} \models \neg\widehat{\phi}$.

We build $\widehat{\sigma}$ in the following way: let x_i be any point in I_i, and define

$$\widehat{\sigma}_i(v_Z) = \begin{cases} \top & \text{if } f_i(x_i) \in R_Z \wedge q_i \in Q_Z \\ \bot & \text{otherwise} \end{cases} ; \quad \widehat{\sigma}_i(v_q) = \begin{cases} \top & \text{if } q_i = q \\ \bot & \text{otherwise} \end{cases} .$$

Clearly $\widehat{\sigma} \not\models \widehat{\phi}$. We have to prove that it is a valid run of D_H.

We do this by proving that it satisfies the conditions given by $Init$, Inv, and $Trans$.

- The $Init$ condition is satisfied by the definition of initial run;
- Condition 0) is satisfied by the definition of $\widehat{\sigma}$;
- Condition 1) is satisfied by the definition of $HLTL$ interpretation of q;
- Condition 2) is satisfied at step i by taking $x = f_i(x_i)$;
- Condition 3) is satisfied at step i: if $loc(\sigma_i) = loc(\sigma_{i+1})$, the precondition is false. If $loc(\sigma_i) \neq loc(\sigma_{i+1})$, we know by definition of run that $(loc(\sigma_i), loc(\sigma_{i+1})) \in E$. Furthermore, $x = f_i(u(I_i))$ and $x' = f_{i+1}(l(I_{i+1}))$ satisfy all the requirements by Definition 3, and the same holds for $x = f_i(x_i)$ and $x' = f_{i+1}(x_{i+1})$ since σ is ground;

– Condition 4) is also satisfied at step i: if $loc(\sigma_i) \neq loc(\sigma_{i+1})$, the precondition is false. Otherwise, by Definition 3, we know that either $u(I_i) \in I_i$ or $l(I_{i+1}) \in I_{i+1}$. We consider the first case (the second is analogue). let $f = f_i \cup f_{i+1}$. Since f is analytic by definition, we know that $f(u(I_i)) \in \overline{f(I_{i+1})}$. Together with groundness of σ, this implies that for every $t \in I_{i+1}$ and for every R_Z such that $f_{i+1}(t) \in R_Z$, $f(u(I_i)) \in \overline{R_Z}$. Therefore $x = f(u(I_i))$ satisfies the existential quantifier in 4).

□

C Data for experimental evaluation

In the tables, we write TO when a timeout occurred (10 hours). Entries in red correspond to cases where the model checker answered "False", hence the algorithm described in Section 3.1 returned "Unknown".

C.1 Linear overtake

We provide in Tables 6-7 the complete data from the experiments. Remarkably, using $ic3$ resulted in a timeout (it took more than 10 hours), while using $ic3-e$ the complete algorithm takes only 6 minutes.

T^{ic3-e}_{TOT}	T^{ic3}_{TOT}	T_{Lyap}	T_{ASJump}	T_{ASReg}	T_{AdReg}	T_{RSOMat}	T_{RSO}	$T_{ic3-e}p1$	$T_{ic3-e}p2$	$T_{ic3-e}p3$	$T_{ic3}p1$	$T_{ic3}p2$	$T_{ic3}p3$	md_{RSO}	ms_{RSO}	ms_{bar}	N_{LTL}	$\lvert\mathcal{Z}\rvert$
359.88	TO	0.45	4.11	1.37	32.85	93.63	4.07	68.94	49.70	6.72	TO	TO	TO	15.50	532.17	68.33	153.00	40.00

Table 6. Data for linear overtake benchmark with images through jump functions.

T^{ic3-e}_{TOT}	T^{ic3}_{TOT}	T_{Lyap}	T_{ASJump}	T_{ASReg}	T_{AdReg}	T_{RSOMat}	T_{RSO}	$T_{ic3-e}p1$	$T_{ic3-e}p2$	$T_{ic3-e}p3$	$T_{ic3}p1$	$T_{ic3}p2$	$T_{ic3}p3$	md_{RSO}	ms_{RSO}	ms_{bar}	N_{LTL}	$\lvert\mathcal{Z}\rvert$
42.73	273.45	0.39	1.66	0.15	8.36	0.00	2.23	3.09	2.63	2.25	192.02	17.18	17.04	0.00	0.00	55.33	92.00	22.00

Table 7. Data for linear overtake benchmark without images through jump functions.

C.2 Multiple choice

Here we list the precise data for the construction of the hybrid system for the benchmark presented in Section 4.2. See also Fig. 9.

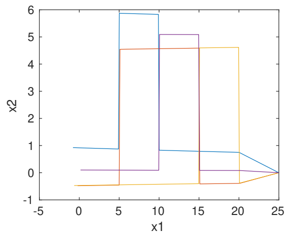

Fig. 7. Four simulations for MC_3.

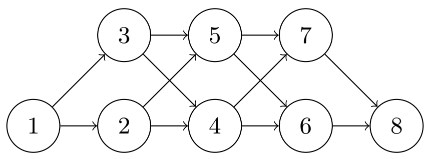

Fig. 8. The scheme of the edges for discrete transitions for MC_n with $n = 3$.

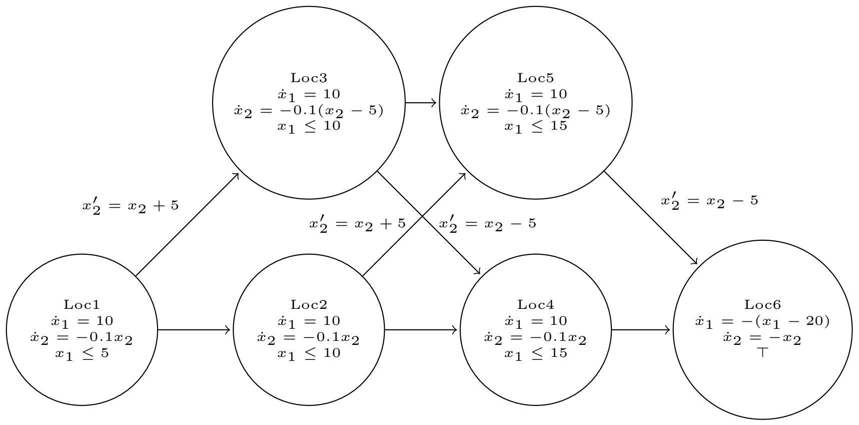
Loc1 $\dot{x}_1 = 10$ $\dot{x}_2 = -0.1x_2$ $x_1 \le 5$
Loc2 $\dot{x}_1 = 10$ $\dot{x}_2 = -0.1x_2$ $x_1 \le 10$
Loc3 $\dot{x}_1 = 10$ $\dot{x}_2 = -0.1(x_2 - 5)$ $x_1 \le 10$
Loc4 $\dot{x}_1 = 10$ $\dot{x}_2 = -0.1x_2$ $x_1 \le 15$
Loc5 $\dot{x}_1 = 10$ $\dot{x}_2 = -0.1(x_2 - 5)$ $x_1 \le 15$
Loc6 $\dot{x}_1 = -(x_1 - 20)$ $\dot{x}_2 = -x_2$ $\top$
$x_2' = x_2 + 5$
$x_2' = x_2 + 5$
$x_2' = x_2 - 5$
$x_2' = x_2 - 5$

Fig. 9. A drawing of the hybrid automaton for the Multiple choice benchmark with $n = 2$. Variables not described in the jump functions stay identical during the jump.

$$X = \{x_1; x_2\}$$
$$Q = \{Loc_1, \dots, Loc_{2n+1}\}$$
$$q_{init} = Loc_1$$
$$init = [-1, 1] \times [-1, 1]$$
$$E = \{(Loc_1, Loc_2), (Loc_1, Loc_3)\} \cup$$
$$\{(Loc_i, Loc_{i+2}), (Loc_i, Loc_{i+3})\}_{i=2,4,\dots,2(n-1)} \cup$$
$$\{(Loc_i, Loc_{i+1}), (Loc_i, Loc_{i+2})\}_{i=3,5,\dots,2n-1} \cup$$
$$\{(Loc_{2n}, Loc_{2n+2}), (Loc_{2n+1}, Loc_{2n+2})\}$$
$$flow(Loc_i) = \begin{cases} \dot{x_1} = 10;\ \dot{x_2} = -0.1(x_2) & \text{if } i = 1 \text{ or } i \text{ is even and } i \neq 2(n+1) \\ \dot{x_1} = 10;\ \dot{x_2} = -0.1(x_2 - 5) & \text{if } i \text{ is odd and } i \neq 1 \end{cases}$$
$$flow(Loc_{2(n+1)}) = \dot{x_1} = -(x_1 - 5(n+2));\ \dot{x_2} = -(x_2)$$
$$inv(Loc_i) = \begin{cases} x_1 \le 5\lfloor \frac{i}{2} \rfloor + 5 & \text{if } i \neq 2n+2 \\ \top & \text{otherwise} \end{cases}$$
$$jump((Loc_1, Loc_2))(x_1, x_2) = (x_1, x_2)$$
$$jump((Loc_1, Loc_3))(x_1, x_2) = (x_1, x_2 + 5)$$
$$jump((Loc_i, Loc_{i+2}))(x_1, x_2) = (x_1, x_2) \text{ for } i = 2, 4, \dots, 2n \text{ and } i = 3, 5, \dots, 2n-1$$
$$jump((Loc_i, Loc_{i+3}))(x_1, x_2) = (x_1, x_2 + 5) \text{ for } i = 2, 4, \dots, 2(n-1)$$
$$jump((Loc_i, Loc_{i+1}))(x_1, x_2) = (x_1, x_2 - 5) \text{ for } i = 3, 5, \dots, 2n+1$$

We provide in Tables 8-9 the complete data from the experiments.

i	T_{TOT}^{ic3-e}	T_{TOT}^{ic3}	T_{Lyap}	T_{ASJump}	T_{ASReg}	T_{AdReg}	T_{RSOMat}	T_{RSO}	T_{ic3-e}	T_{ic3}	md_{RSO}	ms_{RSO}	ms_{bar}	N_{LTL}	$\|\mathcal{Z}\|$
1	330.50	383.03	0.18	0.54	0.22	6.29	282.06	10.37	7.93	30.67	8.10	201.52	106.14	113.00	21.00
2	586.90	680.47	0.24	1.61	0.45	9.62	505.54	20.59	20.98	62.67	8.94	273.53	174.00	203.00	33.00
3	957.21	1081.66	0.29	3.25	0.83	12.71	808.68	34.96	34.07	96.97	9.00	311.97	246.57	293.00	45.00
4	1300.39	1442.51	0.35	5.99	1.44	16.10	1055.80	52.09	77.05	139.62	8.80	341.20	319.14	383.00	57.00
5	1520.82	1691.44	0.41	8.82	2.11	17.02	1223.89	67.18	107.85	172.77	8.95	374.41	373.43	443.00	65.00
6	1931.37	2091.48	0.50	13.45	3.20	20.93	1452.34	87.87	155.19	217.07	8.83	403.63	446.00	533.00	77.00
7	2394.91	2646.79	0.58	20.24	4.72	23.73	1751.96	111.33	236.45	290.69	8.69	431.80	518.57	623.00	89.00
8	2653.55	2999.42	0.69	29.77	6.93	28.16	2019.17	133.52	185.61	360.20	8.47	454.65	583.43	713.00	101.00
9	3513.17	3201.34	0.81	41.37	9.63	31.37	2224.08	157.93	342.37	339.58	8.78	506.49	655.43	803.00	113.00
10	4163.76	3805.34	0.92	54.07	12.51	34.00	2533.06	187.81	575.88	434.95	8.66	531.86	728.14	893.00	125.00
11	4517.33	4587.73	1.06	70.19	16.35	37.82	2774.24	216.23	377.71	545.43	8.66	564.01	800.71	983.00	137.00
12	4998.63	4949.46	1.19	84.24	19.99	41.00	2897.86	242.17	464.05	530.31	8.60	589.04	857.29	1043.00	145.00
13	5823.80	6025.79	1.36	106.54	25.12	45.60	3170.79	275.68	412.51	691.65	8.53	616.37	929.86	1133.00	157.00
14	8376.62	7590.11	1.61	132.15	31.33	49.92	3450.41	311.25	2168.43	1682.44	8.49	645.17	1002.43	1223.00	169.00

Table 8. Computation times for multiple choice benchmark, with images through jump functions.

i	T_{TOT}^{ic3-e}	T_{TOT}^{ic3}	T_{Lyap}	T_{ASJump}	T_{ASReg}	T_{AdReg}	T_{RSOMat}	T_{RSO}	T_{ic3-e}	T_{ic3}	md_{RSO}	ms_{RSO}	ms_{bar}	N_{LTL}	$\|\mathcal{Z}\|$
1	140.06	180.18	0.20	0.16	0.05	1.31	131.10	9.20	2.09	14.14	5.53	60.26	84.00	77.00	9.00
2	226.68	266.70	0.24	0.32	0.07	1.41	188.07	16.07	3.71	24.35	5.36	71.32	113.14	107.00	9.00
3	305.63	355.15	0.29	0.56	0.12	1.63	253.05	25.60	5.56	28.33	5.19	83.48	142.29	137.00	9.00
4	376.97	435.02	0.35	0.97	0.20	1.76	312.32	36.70	6.99	32.66	4.92	92.49	171.43	167.00	9.00
5	437.60	479.21	0.40	1.42	0.28	1.86	345.74	45.87	9.98	33.10	4.73	103.41	192.86	187.00	9.00
6	507.45	568.29	0.48	2.15	0.42	2.03	403.87	58.90	9.76	42.24	4.62	115.30	222.00	217.00	9.00
7	588.89	673.07	0.55	3.17	0.59	2.20	470.19	73.43	13.75	55.21	4.66	131.34	251.14	247.00	9.00
8	691.18	809.66	0.68	4.63	0.82	2.44	539.95	86.53	23.60	89.74	4.44	141.69	278.29	277.00	9.00
9	789.77	907.10	0.79	6.23	1.12	2.64	595.52	101.77	28.95	76.68	4.60	161.69	307.43	307.00	9.00
10	872.77	981.30	0.93	8.29	1.48	2.71	648.55	118.04	21.67	90.52	4.52	173.77	336.57	337.00	9.00
11	1010.48	1073.09	1.07	10.87	1.92	2.95	715.18	136.87	45.22	96.32	4.45	185.79	365.71	367.00	9.00
12	1068.89	1191.35	1.23	13.37	2.46	3.18	749.64	151.87	50.60	143.60	4.37	196.67	387.14	387.00	9.00
13	1212.79	1277.41	1.43	16.70	3.02	3.52	800.85	170.99	86.26	102.63	4.30	207.52	416.29	417.00	9.00
14	1393.54	1451.28	1.65	20.90	3.71	3.69	874.37	192.34	110.12	133.96	4.23	217.87	445.43	447.00	9.00

Table 9. Computation times for multiple choice benchmark, without images through jump functions.

C.3 Navigation benchmark

Hybrid Automaton The set of variables X consists in x_1, x_2, v_1, v_2, where the first two coordinates represent the position and the last two the velocity of the object of interest.

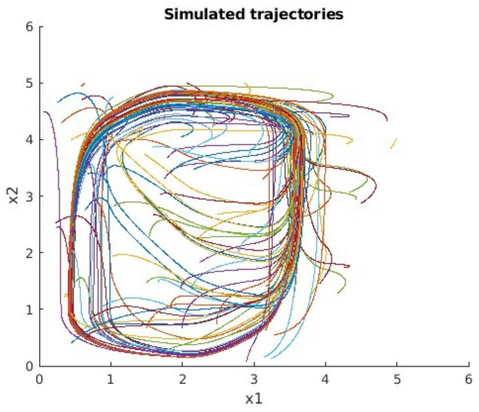

Fig. 10. One hundred trajectories simulated for 100 seconds for the Navigation benchmark, that highlight the possible existence of a loop.

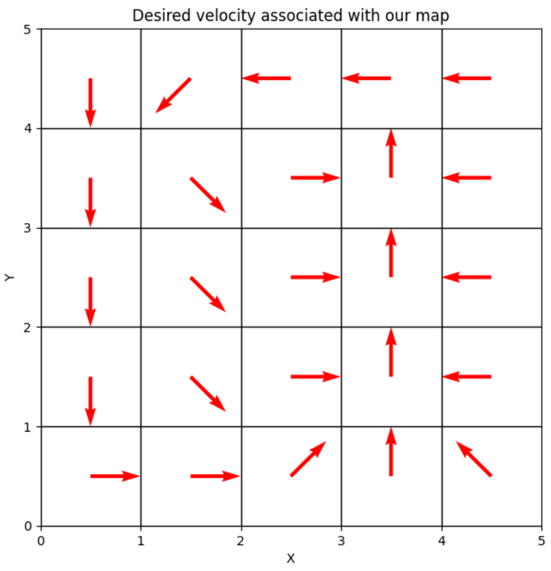

Fig. 11. The desired velocities in each square associated with the map of our instance of the Navigation benchmark.

The square $\{x_1 \in [0,5],\ x_2 \in [0,5]\}$ is partitioned into 25 squares of the form $[k, k+1] \times [h, h+1]$ for $h, k = 0, \ldots, 4$. Each of these squares is associated with an integer $m = 0, \ldots, 7$, that is collected in a matrix called *map*; in our case, the map is

$$\begin{pmatrix} 4\,5\,6\,6\,6 \\ 4\,3\,2\,0\,6 \\ 4\,3\,2\,0\,6 \\ 4\,3\,2\,0\,6 \\ 2\,2\,1\,0\,7 \end{pmatrix}.$$

One associates with each such integer a vector that represents a possible "desired velocity", as $v_m = (\sin(m * \pi/4); \cos(m * \pi/4))$. Let A be the matrix

$$A = \begin{pmatrix} -1.2 & 0.1 \\ 0.1 & -1.2 \end{pmatrix}.$$

The continuous dynamics in a square associated with index m is defined as

$$\dot{x_1} = v_1;\ \dot{x_2} = v_2;\ \dot{\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}} = A\left(\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} - v_m\right).$$

We depict the desired velocity in each square in Figure 11. Intuitively, in each square the velocity tends to converge to the value v_m, while x_1 and x_2 change depending on the instantaneous value of v_1 and v_2. We refer to [FI04] for more details.

Local regions Here we list the precise data for the construction of the regions used for the abstraction for the Navigation benchmark in Section 4.2. We count Locations starting from top-left with Loc_0, moving to right and then bottom until the bottom-right corner, which is Loc_{24}.

$$\begin{aligned}
Z_0 =& ([3.49; 3.55] \times [4.26; 4.32] \times [-0.35; -0.29] \times [0.65; 0.69], \{Loc_3\}) \\
Z_1 =& (x_2 = 1 \wedge \neg (x_1 \in [0.37, 0.46] \wedge v_1 \in [-0.02, -0.002] \wedge v_2 \in [-1.01, 0.994]), \{Loc_{15}\}) \\
Z_2 =& (x_2 = 4 \wedge \neg (x_1 \in [3.565, 3.95] \wedge v_1 \in [-0.02, 0.02] \wedge v_2 \in [0.99, 1.01]), \{Loc_8\}) \\
Z_3 =& ([0.37, 0.46] \times [1, 1] \times [-0.02, -0.002] \times [-1.01, 0.994], \{Loc_{20}\}) \\
Z_4 =& ([3.565, 3.95] \times [4, 4] \times [-0.02, 0.02] \times [0.99, 1.01], \{Loc_3\})
\end{aligned}$$

Notice that regions R_{Z_1} and R_{Z_2} are non-convex, and CORA does not efficiently compute intersections with such regions. For this reason, we split each of them in 6 different regions, obtaining a total of 15 regions.

C.4 Non-linear circular

Here we list the precise data for the construction of the regions used for the abstraction for the Non-linear circular dynamics in Section 4.2.

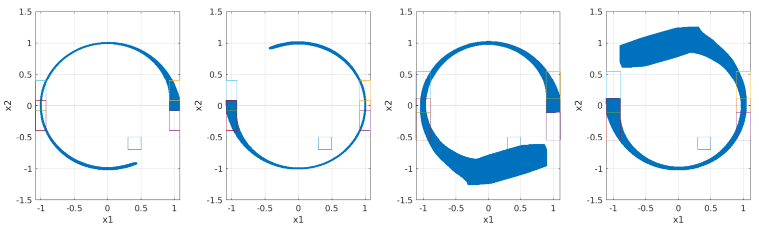

Fig. 12. RSO for $T = 5$ and different values of $\alpha = 0.08, 0.11$ for regions $R_{0,\alpha}$ and $R_{1,\alpha}$.

$$
\begin{aligned}
R_{0,\alpha} =& [1-\alpha, 1+\alpha] \times [0-\alpha, 0+\alpha]\\
R_{2,\alpha} =& [1-\alpha, 1+\alpha] \times [\alpha, 5\alpha]\\
R_{3,\alpha} =& [1-\alpha, 1+\alpha] \times [-5\alpha, -\alpha]\\
R_{4,\alpha} =& [1-\alpha, 1+\alpha] \times [-\alpha, 5\alpha]\\
R_{5,\alpha} =& [1-\alpha, 1+\alpha] \times [-5\alpha, 0+\alpha]\\
R_{6,\alpha} =& [1-\alpha, 1+\alpha] \times [-5\alpha, 5\alpha]\\
R_{1,\alpha} =& [-1-\alpha, -1+\alpha] \times [0-\alpha, 0+\alpha]\\
R_{7,\alpha} =& [-1-\alpha, -1+\alpha] \times [\alpha, 5\alpha]\\
R_{8,\alpha} =& [-1-\alpha, -1+\alpha] \times [-5\alpha, -\alpha]\\
R_{9,\alpha} =& [-1-\alpha, -1+\alpha] \times [-\alpha, 5\alpha]\\
R_{10,\alpha} =& [-1-\alpha, -1+\alpha] \times [-5\alpha, 0+\alpha]\\
R_{11,\alpha} =& [-1-\alpha, -1+\alpha] \times [-5\alpha, 5\alpha]\\
B =& [0.3, 0.5] \times [-0.7, -0.5]
\end{aligned}
$$

Complete results are shown in Table 10.

α	T_{TOT}^{ic3-e}	T_{TOT}^{ic3}	T_{Lyap}	T_{ASJump}	T_{ASReg}	T_{AdReg}	T_{RSOMat}	T_{RSO}	$T_{ic3-e}p_1$	$T_{ic3}p_1$	$T_{ic3-e}p_2$	$T_{ic3}p_2$	md_{RSO}	ms_{RSO}	ms_{bar}	N_{LTL}	$\lvert\mathcal{Z}\rvert$
0.05	182.03	220.00	0.15	0.00	0.04	3.54	107.50	85.64	15.59	25.56	0.70	3.27	12.00	255.67	0.00	6.00	13.00
0.08	116.06	115.70	0.14	0.00	0.04	3.56	31.65	66.53	25.98	21.92	0.49	2.00	12.00	263.00	0.00	2.00	13.00
0.11	103.01	105.23	0.14	0.00	0.04	3.44	32.38	49.42	21.10	21.79	0.41	1.71	12.00	255.00	0.00	2.00	13.00

Table 10. Computation times for nonlinear circular benchmark.

Multilayer control Here we present the complete tables and the images both for the linear and nonlinear case.

i	T_{TOT}^{ic3-e}	T_{TOT}^{ic3}	T_{Lyap}	T_{ASJump}	T_{ASReg}	T_{AdReg}	T_{RSOMat}	T_{RSO}	$T_{ic3-e}p_1$	$T_{ic3}p_1$	$T_{ic3-e}p_2$	$T_{ic3}p_2$	md_{RSO}	ms_{RSO}	ms_{bar}	N_{LTL}	$\|\mathcal{Z}\|$
4	835.34	2067.21	0.42	58.92	5.39	34.11	388.45	18.15	70.35	404.79	93.95	320.04	10.45	497.77	54.00	243.00	90.00
5	3618.56	9796.48	0.78	334.25	29.87	79.64	784.49	38.30	349.10	1770.67	374.34	1288.11	10.46	651.37	59.78	379.00	137.00
6	6715.99	12164.60	0.88	1132.30	78.06	127.44	1379.73	76.12	875.74	2158.84	808.51	2560.31	10.43	801.96	63.94	533.00	194.00
7	17586.29	24945.88	1.26	3751.07	239.99	265.63	2246.24	136.85	2627.76	3939.36	1813.84	3628.16	9.98	923.27	68.92	717.00	261.00

Table 11. Computation times for multilayer control linear benchmark, with images through jump functions.

i	T_{TOT}^{ic3-e}	T_{TOT}^{ic3}	T_{Lyap}	T_{ASJump}	T_{ASReg}	T_{AdReg}	T_{RSOMat}	T_{RSO}	$T_{ic3-e}p_1$	$T_{ic3}p_1$	$T_{ic3-e}p_2$	$T_{ic3}p_2$	md_{RSO}	ms_{RSO}	ms_{bar}	N_{LTL}	$\|\mathcal{Z}\|$
4	23.87	69.80	0.44	2.01	0.09	2.53	0.00	1.53	1.71	13.56	2.12	9.86	0.00	0.00	29.27	45.00	24.00
5	29.88	92.39	0.78	4.65	0.13	3.11	0.00	1.46	1.85	13.45	2.47	21.55	0.00	0.00	28.89	55.00	29.00
6	35.72	124.05	0.84	6.47	0.15	3.44	0.00	1.57	2.51	18.30	3.22	31.71	0.00	0.00	28.62	65.00	34.00
7	46.69	176.11	1.37	11.00	0.19	4.20	0.00	1.69	3.61	45.55	4.56	25.15	0.00	0.00	28.43	75.00	39.00

Table 12. Computation times for multilayer control linear benchmark, without images through jump functions.

i	T_{TOT}^{ic3-e}	T_{TOT}^{ic3}	T_{Lyap}	T_{ASJump}	T_{ASReg}	T_{AdReg}	T_{RSOMat}	T_{RSO}	$T_{ic3-e}p_1$	$T_{ic3}p_1$	$T_{ic3-e}p_2$	$T_{ic3}p_2$	md_{RSO}	ms_{RSO}	ms_{bar}	N_{LTL}	$\|\mathcal{Z}\|$
4	460.37	1230.64	0.56	59.52	5.36	33.13	0.02	252.00	25.67	218.51	23.62	219.38	0.00	0.00	54.00	177.00	90.00
5	1218.42	4889.19	1.57	335.51	30.10	78.18	0.04	385.31	102.21	892.16	107.73	1010.74	0.00	0.00	59.78	271.00	137.00
6	3002.79	12825.42	1.54	1122.59	77.91	125.81	0.06	518.01	300.77	2890.76	376.98	2316.50	0.00	0.00	63.94	373.00	194.00
7	7568.78	28400.18	2.35	3675.47	232.57	223.97	0.10	737.13	785.55	6720.85	785.00	7330.85	0.00	0.00	69.24	495.00	261.00

Table 13. Computation times for multilayer control nonlinear benchmark, with images through jump functions.

i	T_{TOT}^{ic3-e}	T_{TOT}^{ic3}	T_{Lyap}	T_{ASJump}	T_{ASReg}	T_{AdReg}	T_{RSOMat}	T_{RSO}	$T_{ic3-e}p_1$	$T_{ic3}p_1$	$T_{ic3-e}p_2$	$T_{ic3}p_2$	md_{RSO}	ms_{RSO}	ms_{bar}	N_{LTL}	$\|\mathcal{Z}\|$
4	22.54	55.80	0.57	1.96	0.09	2.35	0.00	1.34	1.39	6.94	1.75	13.48	0.00	0.00	29.27	45.00	24.00
5	31.06	94.14	1.54	4.60	0.13	3.05	0.00	1.46	1.91	12.83	2.88	24.44	0.00	0.00	28.89	55.00	29.00
6	37.44	116.60	1.92	6.35	0.15	3.46	0.00	1.57	2.67	18.31	3.73	21.24	0.00	0.00	28.62	65.00	34.00
7	47.29	131.36	2.28	10.85	0.19	4.15	0.00	1.67	3.65	24.56	4.16	23.82	0.00	0.00	28.43	75.00	39.00

Table 14. Computation times for multilayer control nonlinear benchmark, without images through jump functions.

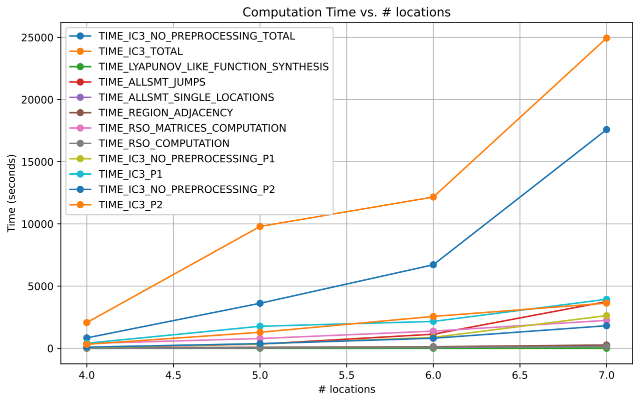

Fig. 13. Computation times for multilayer control benchmark, with images through jump functions.

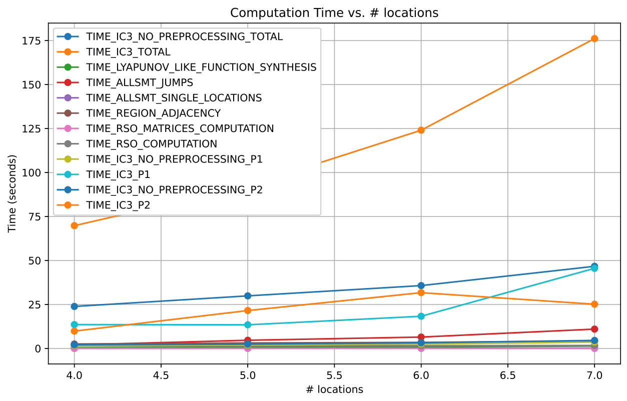

Fig. 14. Computation times for multilayer control benchmark, without images through jump functions.

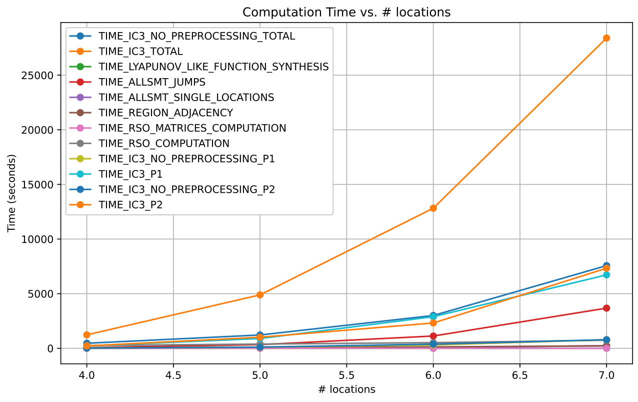

Fig. 15. Computation times for multilayer control benchmark, with images through jump functions.

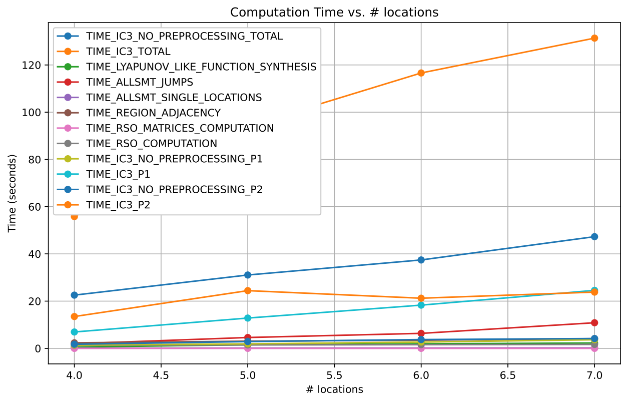

Fig. 16. Computation times for multilayer control benchmark, without images through jump functions.

D Algorithm

Here we show Algorithm 1, that schematize the steps necessary for the computation of the LTL constraints given by the Lyapunov-like functions. Here, $\gamma_{L,Z,q}$ is the LTL property in Theorem 1 and checks are carried out using an SMT solver.

Algorithm 1 Algorithm to obtain constraints from a Lyapunov-like function L. Here, $\nu > 0$ is a chosen treshold and $\gamma_{L,Z,q}$ is the LTL property in Theorem 1. Checks are carried out using an SMT solver.

```
$\gamma_L \leftarrow \top$
for each q in $L_Q$ do
  for each Z in LocalRegions, $q \in Q_Z$ do
    $\varepsilon \leftarrow 10^{-4}$
    if $\models (R_Z \wedge inv(q)) \rightarrow L_F(q) > \varepsilon$ then
      $\gamma_L = \gamma_L \wedge \gamma_{L,Z,q}$
    else if $\models (\neg R_Z \wedge inv(q)) \rightarrow L_F(q) > \varepsilon$ then
      $\gamma_L = \gamma_L \wedge \gamma_{L,\neg Z,q}$
return $\gamma_L$
```

E Appendix for Synthesis of Certificate Functions and Regions

We here elaborate on the synthesis of certificate functions and regions.

Algorithm 2 Algorithm to synthesize Lyapunov-like functions.

```
FunFound $\leftarrow \emptyset$
Candidates $\leftarrow Q \times 2^X$
for $(q, S)$ in Candidates do
  if $var(flow(q)|_S) \subseteq S$ then
    if $flow(q)_S = 0$ has no solution then
      continue
    $z_e \leftarrow$ solution of $flow(q)_S = 0$
    Translate all the system by $z_e$
    if SOS solution for $V > 0, \dot{V} < 0$ for $z \neq 0$ is found and validated then
      $V \leftarrow$ SOS solution
    else if SMT solution for $inv(q) \rightarrow (z \neq 0 \rightarrow (V > 0 \wedge \dot{V} < 0))$ is found then
      $V \leftarrow$ SMT solution
    if $V$ has been found then
      $ValidLocs \leftarrow \{q\}$; Candidates $\leftarrow$ Candidates $\setminus \{(q_2, R), \mid R \subseteq S\}$
      for $q_2 \in Q \setminus ValidLocs$ do
        if $inv(q_2) \rightarrow V > 0 \wedge \dot{V} < 0$ then
          if $guard((q_a, q_b)) \rightarrow V(jump((q_a, q_b))(x)) \leq V(x)$ for $\{q_a, q_b\} \subseteq ValidLocs \cup \{q_2\}$ then
            $ValidLocs \leftarrow ValidLocs \cup \{q_2\}$
            Candidates $\leftarrow$ Candidates $\setminus \{(q_2, R), \mid R \subseteq S\}$
    FunFound $\leftarrow$ FunFound $\cup \{(V(z - z_e), ValidLocs)\}$
return FunFound
```

Lyapunov-like / Descent functions We consider a specific location q, a subset $S = \{z_1, \ldots, z_k\}$ of variables such that the dynamics of the such components is independent from the other components, and a degree d for the candidate function (in our experiments, $d = 2$).

We look for a classic Lyapunov function on the system restricted to the variables S. We start by computing the equilibrium points of the system at location q by solving the system of equations:

$$\dot{S} = flow(q)(S) = 0.$$

Let z_e be a solution. We translate the state variables such that the origin corresponds to an equilibrium point, simplifying subsequent calculations.

The candidate Lyapunov function $V(z)$ is constructed as a polynomial of the specified degree d. For example, if $d = 2$, the quadratic candidate function can be expressed as:

$$V(z) = \sum_{i=1}^{k} \sum_{j=i}^{k} a_{ij} z_i z_j,$$

where n is the number of state variables and a_{ij} are coefficients to be determined.

We then require that:

$$V(z) > 0 \ \wedge \ \dot{V}(z) < 0 \quad \text{for all } 0 \neq z \in inv(q), \tag{3}$$

where here $inv(q)$ is translated by z_e. Recall that $\dot{V}(z)$ can be computed as:

$$\dot{V}(z) = \sum_{i=1}^{k} \frac{\partial V}{\partial z_i}(z) \cdot flow(q)_i,$$

therefore these conditions can be given to an LMI or an SOS solver as described in Section 2.4. Once the function V is sythesized, we validate it through the use of an SMT solver. If either the synthesis or the validation fails, we try to synthesize a Lyapunov function using a symbolic solution. If also this one fails, we move on with a different subset of variables or a different location. Notice than the validity of Conditions (3) implies that the extension of V to the set of all variables is a Lyapunov-like function as in Definition 7 for the system H with $L_Q = \{q\}$. If the Lyapunov-like function is found and validated, we try to expand the set of locations L_Q for which it is valid. To do this, we pick another location q_{new}. We check that the dynamics restricted to S is still independent from the other variables. If this is the case, we verify wheter the couple $(L_f, L_Q \cup \{q_{new}\})$ is a valid Lyapunov-like function by checking using an SMT solver:
- that Conditions (3) hold with $flow(q_{new})$;
- that the condition on the jump in Definition 7 hold for every couple of locations $(q_1, q_2) \in L_Q \cup \{q_{new}\}$.

If this is the case, we can add q_{new} to L_Q and repeat the process for other locations, untill we cannot add any other location.

The synthesis of a descent function is similar to the synthesis of a Lyapunov-like function. The differences consist in: 1) we do not translate the variables to get the equilibrium point in the origin; 2) we select a small $\varepsilon > 0$ and replace Conditions (3) by:

$$\dot{V}(z) < -\varepsilon \quad \text{for all } z \in inv(q).$$

Notice that linear functions appear to be a natural candidate for descent functions, while this is not the case for Lyapunov-like functions.

Region synthesis and Barrier Functions Based on the Lyapunov-like functions that we have found, we create a list of regions that can be of interest. The algorithm to synthesize these regions is described in Algorithm 3. Given a valid

Algorithm 3 Algorithm for the addition of local regions and barrier functions.

```
SublevelLyaps ← $\emptyset$
for $(L_f, L_Q)$ in Lyapunov-like functions do
    $CV_{(L_f,L_Q)} \leftarrow \{0\}$
    for $(R_Z, Q_Z)$ in $\mathcal{Z}$ such that $L_Q \cap Q_Z \neq \emptyset$ do
        $K_{L_f,R_Z} \leftarrow \max_{R_Z} L_f$
        $k_{L_f,R_Z} \leftarrow \min_{R_Z} L_f$
        $isvalid$ ← SMT check for $R \rightarrow L_f > K_{L_f,R_Z}$
        if $isvalid$ then
            $CV_{(L_f,L_Q)} \leftarrow CV_{(L_f,L_Q)} \cup \{K_{L_f,R_Z}\}$
        $isvalid$ ← SMT check for $R \rightarrow L_f < k_{L_f,R_Z}$
        if $isvalid$ then
            $CV_{(L_f,L_Q)} \leftarrow CV_{(L_f,L_Q)} \cup \{k_{L_f,R_Z}\}$
    $CV_{(L_f,L_Q)} \leftarrow sorted(CV_{(L_f,L_Q)})$
    $CV_{(L_f,L_Q)} \leftarrow CV_{(L_f,L_Q)} \cup \{(c1+c2)/2 \mid c1, c2$ consecutive values in $CV_{(L_f,L_Q)}\}$
    SublevelLyaps ← SublevelLyaps $\cup\{(L_f \leq a, L_Q) \mid a \in CV_{(L_f,L_Q)}\}$
BarrierFunctions ← $\emptyset$
ImagesThroughJump ← $\emptyset$
for $(R_Z, Q_Z)$ in SublevelLyaps do
    BarrierFunctions ← BarrierFunctions $\cup\{B((R_Z, Q_Z))\}$
    for $e = (q_1, q_2) \in E$ such that $q_1 \in Q_Z$ do
        ImagesThroughJump ← ImagesThroughJump $\cup\{\mathrm{Im}(R_Z \wedge guard(e))\}$
return SublevelLyaps $\cup$ ImagesThroughJump, BarrierFunctions,
```

Lyapunov-like function (L_f, L_Q), for each local region $(R_Z, Q_Z) \in \mathcal{Z}$ such that $L_Q \cap Q_Z \neq \emptyset$ we approximate (using convex optimization) two values:

$$K_{L_f,R} = \max_R L_f \qquad k_{L_f,R} = \min_R L_f.$$

We then check with an SMT solver that $R \rightarrow L_f > K_{L_f,R}$ (resp., $R \rightarrow L_f < k_{L_f,R}$). If we are able to effectively synthesize and validate either of these two values, we add it to a list of *critical values* $CV_{(L_f,L_Q)}$ for (L_f, L_Q). At the end, we add to this list the value 0 and the mid-points of two consecutive critical values. The possible local regions that can be of intereset are the sublevels of the Lyapunov-like function given by these values:

$$\{(L_f \leq a, L_Q) \mid a \in CV_{(L_f,L_Q)}\}.$$

These local regions are added to the list of local regions used in the abstraction. For each synthesized sublevel of Lyapunov-like function (R_Z, Q_Z)

$$(L_f \leq a, L_Q)$$

we add the *associated barrier function*

$$B((R_Z, Q_Z)) = (L_f - a,\ L_{Q_i})$$

that is a valid barrier certificate and a witness that once a sublevel is reached, the system will not leave it. As a final step, for each local region $(L_f \leq a, L_Q)$ and each $e = (q_1, q_2) \in E$ such that $q_1 \in L_Q$, we compute the image of $(L_f \leq a) \cap guard(e)$ through $jump(e)$, and add it to the list of local regions the ones that are non-empty.

Note that the computation of an image consists in general of a quantifier elimination over the reals. Since this process can be computationally very heavy, we compute such images only when the jump is given by an invertible map (hence, the computation consists in a substitution) or when the jump consists in a constant map (*e.g.:* $x' = 5$). This is always the case in our benchmarks.