Chapter 30: Verification of Hybrid Systems

Laurent Doyen, Goran Frehse, George J. Pappas, and André Platzer

Abstract Hybrid systems are models with joint discrete and continuous behavior. They occur frequently in safety-critical applications in various domains such as health care, transportation, and robotics, as a result of interactions between a digital controller and a physical environment. They also have relevance in other areas such as systems biology, in which the discrete dynamics arises as an abstraction of fast continuous processes. One of the prominent models is that of *hybrid automata*, where differential equations are associated with each node, and jump constraints such as guards and resets are associated with each edge.

In this chapter, we focus on the problem of model checking of hybrid automata against reachability and invariance properties, enabling the techniques for the verification of general temporal logic specifications. We review the main decidability results for hybrid automata, and since model-checking is in general undecidable, we present three complementary analysis approaches based on symbolic representations, abstraction, and logic. In particular, we illustrate polyhedron-based reachability analysis, finite quotients, abstraction refinement techniques, and logic-based verification. We survey important tools and application domains of successful hybrid system verification in this vibrant area of research.

Laurent Doyen
LSV, CNRS & ENS Paris-Saclay, Cachan, France
e-mail: doyen@lsv.ens-cachan.fr

Goran Frehse
Verimag, Université Joseph Fourier - Grenoble 1, France
e-mail: frehse@imag.fr

George J. Pappas
Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA
e-mail: pappasg@ee.upenn.edu

André Platzer
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA
e-mail: aplatzer@cs.cmu.edu

© Springer International Publishing AG, part of Springer Nature 2018
E.M. Clarke et al. (eds.): *Handbook of Model Checking*,
DOI: 10.1007/978-3-319-10575-8_30

Contents

Chapter 30: Verification of Hybrid Systems 1
Laurent Doyen, Goran Frehse, George J. Pappas, and André Platzer
1 Introduction 3
2 Basic Definitions 5
2.1 Predicates 6
2.2 Hybrid Automata 7
3 Decidability and Undecidability Results 10
4 Set-based Reachability Analysis 14
4.1 Reachability Algorithm 14
4.2 Piecewise Constant Dynamics 16
4.3 Piecewise Affine Dynamics 20
4.4 Nonlinear Dynamics 30
5 Abstraction-based Verification 31
5.1 Discrete Abstractions 32
5.2 Phase-portrait approximation 33
5.3 Predicate Abstractions 35
5.4 Abstraction Refinement 38
5.5 Approximate Bisimulations 39
6 Logic-based Verification 40
6.1 Polynomial Barrier Certificates 43
6.2 Equational Certificates 44
6.3 Differential Invariants and Logical Certificates 46
7 Verification Tools 53
References 57

1 Introduction

Information technology (IT) has dramatically changed our lives. The first revolution in information technology led to the birth of the computer. The second information technology revolution led to the creation of the Internet, connecting computing around the world and resulting in the hyper-connected world that we live in. The third revolution that is now taking place is connecting all the computational power to the physical world. Computing powerhouses, such as Intel, are investing in wearable computing and smart watches, Google is invested in self-driving cars and bought Nest, the makers of a learning thermostat, thereby connecting Google to building control and energy markets, and Amazon is investing in robotics and unmanned aerial vehicles. Similarly the most significant innovation in automotive companies recently came from software-intensive car technology, leading from adaptive cruise control to driverless cars. There is a plethora of novel medical devices, either wearable or implantable, that sense patient vitals and use computer algorithms to diagnose medical conditions or even perform life-critical functions. The fundamental aspect of this third revolution is the fusion of IT with physical devices that interact with the physical world.

The marriage of IT with the physical world is known as embedded computing as it consists of computing that is embedded and tightly interacts with the physical world. A major modeling challenge is how to formally capture the interaction between computing and physics so that we can *reason about the effect of physics on computing and vice versa*. This led to the development of *hybrid systems* [8, 132, 34, 91, 35], where both discrete and continuous behaviors of the system are important. Hybrid systems grew out of the necessity to enrich purely digital models of computing with analog models of physics. As a result, hybrid systems contain both digital models of computing (such as automata or programs) as well as analog elements (such as differential equations) integrated in a way that one can model many embedded computing applications.

The need for formal models of hybrid systems arises from the fact many embedded computing problems are safety-critical. They arise in collision avoidance protocols in air traffic control [175, 174, 113, 31, 129, 176, 177, 149, 100], cruising controllers for automotive vehicles [46, 168, 101, 164, 24, 62, 53, 114], obstacle avoidance algorithms for autonomous ground robots [182, 130], and software-controlled medical devices that actively regulate life-critical functions or help surgeons with surgical robotic systems [104]. Therefore there is not only a need for formal models of embedded computing, but also for rigorous verification approaches to guarantee that the embedded computation, as modeled by a hybrid system, is formally safe. This has resulted in the development of a new paradigm within the formal methods community, namely the formal verification of hybrid models of embedded computing.

There is a range of formal models for hybrid systems [167, 120, 131, 132, 134, 8, 27, 35, 91, 118, 52, 29, 179, 180, 135, 143], each with different advantages for different purposes. This chapter focuses on *hybrid automata* [8, 91], because they directly generalize the timed automata that have been considered in Chapter 27. The basic idea in hybrid automata is to associate differential equations with the nodes of an automaton. The automaton structure defines how and under which condition the system switches between the various differential equations and what happens to the state if they switch. Timed automata, which are discussed in Chapter 27, are a special case of hybrid automata, where all differential equations are of the form $\dot{x} = 1$ such that x is a clock variable measuring the progress of time and additional linearity assumptions are met for the switching conditions. Timed automata are an interesting subclass of hybrid automata, because reachability is decidable in this subclass. For systems with more general continuous dynamics, e.g., moving, acceleration, or curving, however, timed automata are not sufficient, and hybrid systems models are needed instead.

In this chapter, we give a survey of model checking techniques for hybrid systems with an emphasis on the handling of continuous dynamics. It has been proved that continuous dynamics verification is the most fundamental question in hybrid systems verification [135, 141], because discrete dynamics can be verified exactly as good as continuous dynamics. Discrete systems have already been addressed in the other chapters of this handbook in great detail.

In this section, we survey a set of complementary verification techniques for hybrid systems, including explicit-state reachability computations with termination criteria like bounded-horizon (Sect. 4, which is related to Chap. **??**), abstraction techniques and abstraction-refinement loops (Sect. 5, also see Chap. **??**), and logic-based verification approaches (Sect. 6, which is related to Chaps. **??**, **??**, **??**, and **??**). Other surveys of several aspects of hybrid systems can be found in the literature [12, 172, 39, 80, 7, 163, 171, 103, 143]. A control-theoretic view on hybrid systems verification has been reported in a book by Tabuada [165]. A logic and proofs view on hybrid systems verification can be found in a book by one of the authors [137]. Introductions to embedded systems from a cyber-physical systems perspective have been reported in the literature [122, 111] and in university courses.

Hybrid systems have become a very active and successful area of research with a vibrant community. Giving a complete overview of all relevant approaches is impossible in this chapter. This chapter strives to focus on giving an overview of some of the most important representative classes of techniques. By their very nature, hybrid systems tend to be mathematically demanding, but can also be exceedingly beautiful. The broad applicability and scope of the resulting hybrid systems analysis techniques make hybrid systems a very rewarding area of science with the potential of significant impact on practical applications.

2 Basic Definitions

Hybrid systems combine discrete evolutions (namely, mode changes and variable updates) and continuous evolutions through variables whose dynamics is governed by differential equations. Hybrid system models have been introduced to deal with such systems in a uniform way [167, 120, 131, 132, 134, 8, 91, 27, 35, 118, 52, 29, 179, 180, 135, 136]. The original definitions are very general. In this chapter, we focus on subclasses of particular interest. Timed automata are an important class of hybrid automata for which safety verification is decidable (see Chap. **??**). When continuous variables are subject to rectangular flow constraints, that is constraints of the form $\dot{x} \in [a,b]$, hybrid automata are called *rectangular*. For that subclass of hybrid automata, there exists a reasonably efficient algorithm to compute the image of a (simple) set. Based on this algorithm, there exists an iterative method that computes the exact set of reachable states when it terminates. This semi-algorithm can be used to establish or refute *safety properties*. On the other hand, if the evolution of the continuous variables is subject to more complicated flow constraints, for example affine dynamics like $\dot{x} = 3x - y$, computing the flow successor is much more difficult and only approximate methods are known.

2.1 Predicates

Let $X = \{x_1, \ldots, x_n\}$ be a finite set of variables. Given a valuation $v : X \to \mathbb{R}$ and $Y \subseteq X$, define $v_{|Y} : Y \to \mathbb{R}$ by $v_{|Y}(x) = v(x)$ for every $x \in Y$.

Definition 1 (Polynomial term). A *polynomial term* over a finite set of variables $X = \{x_1, \ldots, x_n\}$ is an expression of the form $y \equiv \sum_{i \in \mathbb{N}^n} a_i x_1^{i_1} \ldots x_n^{i_n}$ where $a_i \in \mathbb{Q}$ ($i = (i_1, \ldots, i_n) \in \mathbb{N}^n$) are rational constants and almost all a_i are zero. Given a valuation v over X, we write $[\![y]\!]_v$ for the real number $\sum_{i \in \mathbb{N}^n} a_i v(x_1)^{i_1} \ldots v(x_n)^{i_n}$ obtained by evaluating the polynomial term at v. We denote by $\mathsf{PTerm}(X)$ the set of all polynomial terms over the variables X.

Definition 2 (Polynomial constraint). A *polynomial constraint* over X (also known as semi-algebraic constraint) is a finite formula φ defined by the following grammar rule:

$$\varphi ::= \theta \bowtie 0 \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$

where $\theta \in \mathsf{PTerm}(X)$ and $\bowtie \in \{<, \leq, =, >, \geq\}$. We denote by $\mathsf{PConstr}(X)$ the class of polynomial constraints over the set of variables X.

Definition 3. Given a valuation $v : X \to \mathbb{R}$ and a polynomial constraint $\varphi \in \mathsf{PConstr}(X)$, we write $v \models \varphi$ and say that v *satisfies* φ, which we define inductively as:

- $v \models \theta \bowtie 0$ if $[\![\theta]\!]_v \bowtie 0$,
- $v \models \varphi_1 \wedge \varphi_2$ if $v \models \varphi_1$ and $v \models \varphi_2$,
- $v \models \varphi_1 \vee \varphi_2$ if $v \models \varphi_1$ or $v \models \varphi_2$.

We also write $v \in [\![\varphi]\!]$ when $v \models \varphi$. If $v : X \to \mathbb{R}$ and $w : Y \to \mathbb{R}$ are two valuations for disjoint sets of variables X, Y (with $X \cap Y = \varnothing$), we also write $(v, w) \in [\![\varphi]\!]$ when $u \models \varphi$ where $u : X \cup Y \to \mathbb{R}$ is defined such that $u_{|X} = v$ and $u_{|Y} = w$.

Important special cases of polynomial constraints are linear constraints. The set of solutions of a linear constraint describes a set of polyhedra. This geometric interpretation is sometimes used for model checking since image computations of polyhedra can be quite efficient.

Definition 4 (Linear constraint). A *linear term* is a polynomial term of the form $y \equiv a_0 + \sum_{x_i \in X} a_i x_i$ with $a_i \in \mathbb{Q}$. We denote the set of all linear terms over X by

$\mathsf{LTerm}(X)$. A *linear constraint* is a polynomial constraint where all terms are linear. It is called *conjunctive* if it does not contain any disjunctions. We denote by $\mathsf{LConstr}(X)$ the class of linear constraints over X and by $\mathsf{LConstr}_c(X)$ the class of conjunctive linear constraints. The constraints true and false are defined as abbreviations in a standard way.

Definition 5 (Polyhedron). A set of valuations that can be defined by a conjunctive linear constraint is called a *polyhedron*, and a closed and bounded polyhedron is called a *polytope*. We denote a polyhedron in its *constraint representation* as

$$P = \left\{ x \;\middle|\; \bigwedge_{i=0}^{m} a_i^{\mathsf{T}} x \bowtie_i b_i \right\}, \text{ with } \bowtie_i, \in \{<, \leq, =, >, \geq\},^1$$

where the $a_i \in \mathbb{Q}^n$ are called *facet normals* and the $b_i \in \mathbb{Q}^n$ constants. In vector-matrix notation, this corresponds to

$$P = \left\{ x \;\middle|\; Ax \bowtie b \right\}, \text{ with } A = \begin{pmatrix} a_1^{\mathsf{T}} \\ \vdots \\ a_m^{\mathsf{T}} \end{pmatrix}, \bowtie = \begin{pmatrix} \bowtie_1 \\ \vdots \\ \bowtie_m \end{pmatrix}, b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}.$$

A closed polyhedron $P \subseteq \mathbb{R}^n$ can be represented by a pair (V, R), called the *generators* of P, where $V \subseteq \mathbb{Q}^n$ is a finite set of *vertices*, and $R \subseteq \mathbb{Q}^n$ is a finite set of *rays*, with:

$$P = \left\{ \sum_{v_i \in V} \lambda_i \cdot v_i + \sum_{r_j \in R} \mu_j \cdot r_j \;\middle|\; \lambda_i \geq 0, \mu_j \geq 0, \sum_i \lambda_i = 1 \right\}.$$

The representation can be extended with *closure points* to deal with non-closed polyhedra [22].

There are algorithms for transforming one representation into the other, namely the Fourier-Motzkin procedure (or quantifier elimination) for computing the system of inequalities from the generators [55, 63], and Chernikova's algorithm for computing the generators from a set of predicates [44].

2.2 *Hybrid Automata*

We define hybrid automata with polynomial dynamics [6, 96].

Definition 6 (Hybrid automaton with polynomial dynamics). A *hybrid automaton H with polynomial dynamics* is a tuple

$$\langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$$

1 $x^{\mathsf{T}} y = \sum_{i=1}^{n} x_i y_i$ is the scalar product of n-dimensional vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$.

where:

- $\mathsf{Loc} = \{\ell_1, \dots, \ell_m\}$ is a finite set of *locations*;
- Lab is a finite set of *labels*, including the *silent* label τ;
- $\mathsf{Edg} \subseteq \mathsf{Loc} \times \mathsf{Lab} \times \mathsf{Loc}$ is a finite set of *edges*;
- $X = \{x_1, \dots, x_n\}$ is a finite set of *variables*;
- $\mathsf{Init} : \mathsf{Loc} \to \mathsf{PConstr}(X)$ gives the *initial condition* $\mathsf{Init}(\ell)$ of location ℓ. The automaton can start in ℓ with an initial valuation v lying in $[\![\mathsf{Init}(\ell)]\!]$;
- $\mathsf{Inv} : \mathsf{Loc} \to \mathsf{PConstr}(X)$ gives the *evolution domain restriction* $\mathsf{Inv}(\ell)$ (also called invariant) of location ℓ. The automaton can stay in ℓ as long as the values of its variables lie in $[\![\mathsf{Inv}(\ell)]\!]$;
- $\mathsf{Flow} : \mathsf{Loc} \to \mathsf{PConstr}(X \cup \dot{X})$ is the *flow constraint*, which constrains the evolution of the variables in each location. In a location ℓ, if the valuation of the variables is v_0 at time $t = 0$, then at time $t \geq 0$, the value of the variables is $\phi(t)$ where $\phi(t) : \mathbb{R} \to \mathbb{R}^X$ is such that the flow relation $\mathsf{Flow}(\ell)(\phi(t), \dot{\phi}(t))$ holds for the flow $\phi(t)$ and its time-derivative $\dot{\phi}(t)$, and $\phi(0) = v_0$. [2]
- $\mathsf{Jump} : \mathsf{Edg} \to \mathsf{PConstr}(X \cup X^+)$ with $X^+ = \{x_1^+, \dots, x_n^+\}$ gives the *jump condition* $\mathsf{Jump}(e)$ of edge e. The variables in X^+ refer to the updated values of the variables after the edge has been traversed. Jump conditions are often conjunctions of a guard and a reset constraint. There, the constraints purely on variables in X are called *guards*, and the constraints that describe variables in X^+ in terms of variables in X are called *updates or resets*.
- $\mathsf{Final} : \mathsf{Loc} \to \mathsf{PConstr}(X)$ gives the *final condition* $\mathsf{Final}(\ell)$ of location ℓ. Depending on the analysis question at hand, final conditions can either specify the unsafe states of the system or the desired states of the system.

The labels on edges can be used to synchronize hybrid automata in a compositional design. In the rest of this chapter, we assume that a single automaton is to be analyzed.

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

[2] Note that the semantics of flow constraints requires some attention, see differential-algebraic constraints [136].

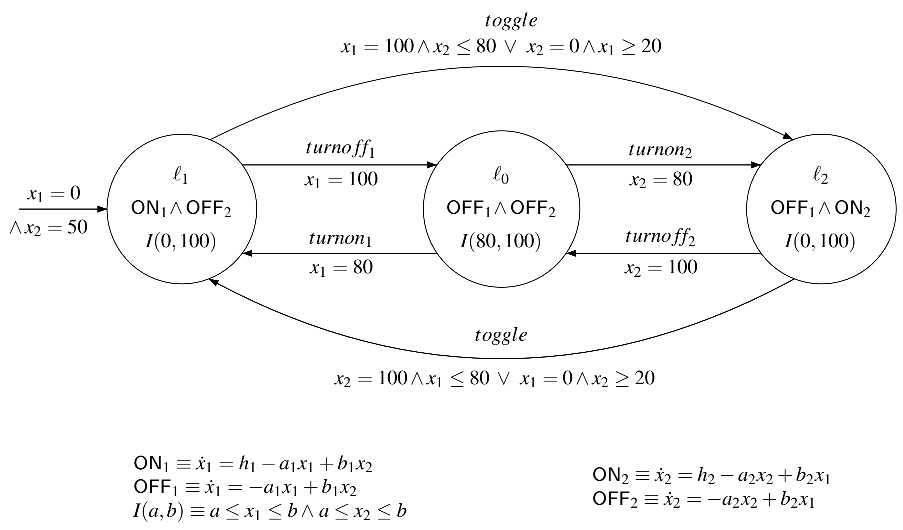

toggle
$x_1 = 100 \wedge x_2 \leq 80 \vee x_2 = 0 \wedge x_1 \geq 20$
turnoff$_1$
$x_1 = 100$
turnon$_2$
$x_2 = 80$
$x_1 = 0$
$\wedge x_2 = 50$
ℓ_1
$\mathsf{ON}_1 \wedge \mathsf{OFF}_2$
$I(0,100)$
ℓ_0
$\mathsf{OFF}_1 \wedge \mathsf{OFF}_2$
$I(80,100)$
ℓ_2
$\mathsf{OFF}_1 \wedge \mathsf{ON}_2$
$I(0,100)$
turnon$_1$
$x_1 = 80$
turnoff$_2$
$x_2 = 100$
toggle
$x_2 = 100 \wedge x_1 \leq 80 \vee x_1 = 0 \wedge x_2 \geq 20$
$\mathsf{ON}_1 \equiv \dot{x}_1 = h_1 - a_1x_1 + b_1x_2$
$\mathsf{OFF}_1 \equiv \dot{x}_1 = -a_1x_1 + b_1x_2$
$I(a,b) \equiv a \leq x_1 \leq b \wedge a \leq x_2 \leq b$
$\mathsf{ON}_2 \equiv \dot{x}_2 = h_2 - a_2x_2 + b_2x_1$
$\mathsf{OFF}_2 \equiv \dot{x}_2 = -a_2x_2 + b_2x_1$

Fig. 1 A shared gas-burner.

100 degrees. Since the temperature of tank 2 is below 80 degrees, the automaton takes edge *toggle* to location ℓ_2. Note that edge *turnoff*$_1$ can not be taken since the evolution domain of the target location is not satisfied by x_2. In location ℓ_2, the burner heats up tank 2 until it reaches 100 degrees. Since x_1 is still above 80 degrees, the automaton takes edge *turnoff*$_2$ to location ℓ_0, where the burner is off. It briefly remains until x_1 falls to 80 degrees, at which point it takes edge *turnon*$_1$ to location ℓ_1, where the burner heats up tank 1. The automaton converges towards a limit cycle of heating tank 1, heating tank 2, and briefly turning off the burner.

The definitions above define what a hybrid automaton consists of (flows, jumps, initial regions, ...) but they do not specify the behavior of a hybrid automaton or how its state evolves over time. This is the purpose of defining a semantics for hybrid automata by providing a transition system for each hybrid automaton.

Definition 7 (Semantics of hybrid automata). The *semantics* of a hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$ is the transition system $[\![H]\!] = \langle S, S_0, S_f, \Sigma, \rightarrow \rangle$ where $S = \{(\ell, v) \in \mathsf{Loc} \times \mathbb{R}^X \mid v \in [\![\mathsf{Inv}(\ell)]\!]\}$ is the *state space*, $S_0 = \{(\ell, v) \in S \mid v \in [\![\mathsf{Init}(\ell)]\!]\}$ is the *initial space*, $S_f = \{(\ell, v) \in S \mid v \in [\![\mathsf{Final}(\ell)]\!]\}$ is the *final space*, the actions are $\Sigma = \mathsf{Lab} \cup \{\mathsf{time}\}$ (we assume that $\mathsf{time} \notin \mathsf{Lab}$) and the transition relation $\rightarrow$ contains all the tuples $((\ell, v), \sigma, (k, w))$ such that:

- (discrete transition) either there exists $e = (\ell, \sigma, k) \in \mathsf{Edg}$ such that $(v, w) \in [\![\mathsf{Jump}(e)]\!]$, or

- (continuous transition) $\ell = k$, $\sigma = \mathsf{time}$ and there exists an $r \in \mathbb{R}^{\geq 0}$ and a continuously differentiable function $\xi : [0,r] \to \mathbb{R}^X$ such that $\xi(0) = v$, $\xi(r) = w$ and $(\xi(t), \dot{\xi}(t)) \in [\![\mathsf{Flow}(\ell)]\!]$ for all $t \in [0,r]$ and $\xi(t) \in [\![\mathsf{Inv}(\ell)]\!]$ for all $t \in [0,r]$. We call ξ a *trajectory* from v to w. We also write $(\ell, v) \xrightarrow{r} (k, w)$ to emphasize that the continuous transition is of duration r. Usually $\mathsf{Flow}(\ell)$ is a differential equation, in which case ξ is a solution of that differential equation.

We write $(\ell, v) \xrightarrow{\sigma} (k, w)$ if $\to$ contains the tuple $((\ell, v), \sigma, (k, w))$.

A state $q = (\ell, v) \in S$ is *reachable* if there exists a finite path $q_0\sigma_0q_1\sigma_1 \ldots \sigma_{n-1}q_n$ where $q_0 \in S_0$, $q = q_n$, and $(q_i, \sigma_i, q_{i+1}) \in \to$ for all $0 \leq i < n$. This path generates the word $\bar{\sigma} = \sigma_0\sigma_1 \ldots \sigma_{n-1} \in \Sigma^*$. If $q \in S_f$ is final, we say that the word $\bar{\sigma}$ is accepted by H. The set of words that are accepted by H is the *language* of H, denoted $\mathcal{L}(H)$. The set of reachable states of $[\![H]\!]$ is denoted by $\mathsf{Reach}([\![H]\!])$. The transition system $[\![H]\!]$ is *safe* if $\mathsf{Reach}([\![H]\!]) \cap S_f = \varnothing$.

Safety verification problem. Many verification problems for hybrid systems reduce to the *safety problem* for hybrid automata.

Definition 8 (Safety verification problem for hybrid automata). Given a hybrid automaton H, the *safety verification problem for hybrid automata* asks whether $[\![H]\!]$ is safe.

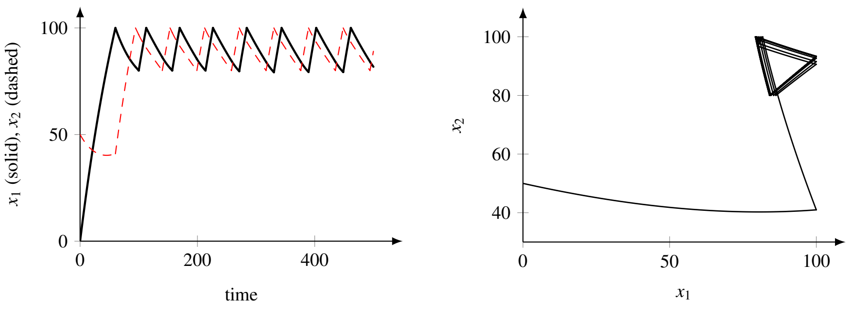
A *parameter* in a hybrid automaton is a variable which has first derivative 0 in every location and is never modified by discrete transitions. The *parametric safety verification problem for hybrid automata* asks, given a hybrid automaton H and a parameter p in H, whether there exists a value $v_p \in \mathbb{R}$ such that $[\![H_{p=v_p}]\!]$ is safe, where $H_{p=v_p}$ is obtained by replacing every constraint φ in H by $\varphi \wedge (p = v_p)$.

Remark There would be no loss of generality in assuming that there is a location ℓ_{bad} such that $\mathsf{Final}(\ell_{\mathsf{bad}}) = \mathsf{true}$ and $\mathsf{Final}(\ell) = \mathsf{false}$ for all $\ell \neq \ell_{\mathsf{bad}}$. Indeed, it suffices to add transitions $e_\ell = (\ell, \sigma, \ell_{\mathsf{bad}})$ with $\mathsf{Jump}(e_\ell) = \mathsf{Final}(\ell)$ for each $\ell \in \mathsf{Loc}$.

3 Decidability and Undecidability Results

We review the main important results about the decidability of the safety verification problem for subclasses of hybrid automata. References are given where details and proofs can be found.

Safety verification problem. The safety verification problem is decidable only for restricted classes of hybrid automata. The main classes for which safety verification is decidable are timed automata (see Chap. **??**), initialized rectangular automata, and o-minimal hybrid automata [108]. The safety verification problem is undecidable already for the class of rectangular hybrid automata (and therefore also for linear, and affine hybrid automata).



(a) x_1, x_2 over time (x_1 solid, x_2 dashed) (b) reachable values of variables x_1 and x_2

Fig. 2 The evolution of the continuous variables of the shared gas burner example, starting from location ℓ_1 with initial values $x_1 = 0$ and $x_2 = 50$

A rectangular predicate over X is an expression of the form $a \prec x \prec b$ where $x \in X$, $\prec \in \{\leq, <\}$ and $a \leq b$ define a nonempty (possibly unbounded) interval with endpoints $a, b \in \mathbb{Q} \cup \{-\infty, \infty\}$. *Rectangular hybrid automata* are hybrid automata where (i) the flow constraint in each location ℓ is a conjunction of rectangular predicates over $\dot{X}$, (ii) the initial, final, and evolution domain conditions are conjunctions of rectangular predicates over X, and (iii) the jump condition of every edge is a conjunction of rectangular predicates over X^+ and expressions of the form $x^+ = x$ for $x \in X$. A hybrid automaton is *initialized* if for every edge $e = (\ell, \sigma, k)$ and for every variable x, if we have $\{v(\dot{x}) \mid v \in [\![\mathsf{Flow}(\ell)]\!]\} \neq \{v(\dot{x}) \mid v \in [\![\mathsf{Flow}(k)]\!]\}$, then the set $\mathsf{update}_e^x(v) = \{w(x^+) \mid (v, w) \in [\![\mathsf{Jump}(e)]\!]\}$ does not depend on the valuation v (i.e., $\mathsf{update}_e^x(v) = \mathsf{update}_e^x(v')$ for all valuations v, v'). In words, whenever the flow condition changes for a variable x by a discrete transition e, then this variable is (nondeterministically) reinitialized to a new value in update_e^x that is independent of the previous value.

The following decidability result is obtained by a translation of initialized rectangular hybrid automata to timed automata, preserving safety (see also Section 5.1).

Theorem 1 ([96]). *The safety verification problem is decidable for initialized rectangular hybrid automata (and therefore also for timed automata).*

The safety verification problem remains decidable for various extensions of timed automata. For instance, if *diagonal constraints* of the form $x - y \bowtie c$ for $x, y \in X, \bowtie \in \{<, \leq, =, >, \geq\}$, and $c \in \mathbb{Q}$ are allowed in guards, or if *assignments* of the form $x^+ = y$ are allowed in updates, then the safety verification problem is still decidable [11, 33]. The decidability result for safety verification, useful for model-checking, can be extended to the controller-synthesis problem, solved as a game. We refer to Chap. **??** for games on timed automata, and mention the decidability of discrete-time control for rectangular hybrid automata [95].

The safety verification problem becomes undecidable for automata with rectangular flow constraints.

Theorem 2 ([96]). *The safety verification problem is undecidable for rectangular hybrid automata (and therefore also for linear, and affine hybrid automata).*

Note that the class of initialized rectangular hybrid automata (for which safety verification is decidable) have a finite language-equivalence quotient [156, 94]. The special case of initialized rectangular hybrid automata with only two variables even has a finite similarity quotient [94], and the class of timed automata has finite bisimilarity quotient (see also Chap. **??**).

The result of Theorem 2 has been refined in several directions [96]. The problem is undecidable even if there is a single variable x with two different slopes, i.e. there exists $k_1, k_2 \in \mathbb{Q}$ with $k_1 \neq k_2$ such that in every location ℓ, either $\mathsf{Flow}(\ell)$ implies $\dot{x} = k_1$, or $\mathsf{Flow}(\ell)$ implies $\dot{x} = k_2$. The undecidability result holds for all fixed rational constants $k_1 \neq k_2$. The problem is also undecidable if diagonal constraints or assignments of the form $x^+ = y$ are allowed, and one variable has slope $k \neq 1$. There are extremely simple classes of hybrid systems, stopwatch automata, i.e. timed automata with only differential equations of the form $\dot{x} = 1$ and $\dot{x} = 0$, that are already undecidable [40] (see also Chap. **??**). The variant of time-bounded safety verification asks, given a time bound T whether there exists a final state reachable within a total duration of T time units. This problem is also undecidable for general rectangular hybrid automata, but it is decidable for a larger class than plain safety verification, namely for rectangular hybrid automata with monotone dynamics (the rate of every variable is either always non-negative, or always non-positive [36]).

Note that while differential equations define single continuous executions, the safety verification problem has been considered under various perturbed semantics with finite precision where drifting executions or tubes of executions are considered. It turns out that the undecidability result of Theorem 2 is mostly robust [97], but some decidability results can be obtained [155, 64, 57, 23].

Systems between timed and hybrid automata may remain decidable, e.g., weighted timed automata [15, 25]. Even systems with piecewise constant derivatives quickly become undecidable for dimension 3 [21]. On the other hand, if the discrete and the continuous parts of a hybrid system are completely independent of each other, the system falls apart into separate continuous systems, so that reachability becomes decidable for certain classes of linear differential equations [108].

Parametric safety verification problem. If parameters are allowed only in the jump conditions of the edges, then it can be shown that the parametric safety verification problem is decidable for timed automata with one clock [14, 124], and undecidable for timed automata with one parameter and (i) three clocks (all of which being possibly constrained by the parameter) [124], or (ii) four clocks, but only one is compared with the parameter [124]. These undecidability results require the use of equalities in jump conditions. An undecidability result is known for *open* timed automata (in which all guards are open sets, thus forbidding equality constraints) with two parameters and five clocks (among which two are compared with the parameters) [59]. If parameters are allowed in the flow constraints, then it can be shown that the parametric safety verification problem is undecidable for rectangular automata with three variables and one parameter [181].

Computability and Polynomial Constraints. A frequent misconception about the definition of hybrid automata is that they should allow an arbitrary subset $\mathsf{Init}(\ell) \subseteq \mathbb{R}^n$ of the real numbers as initial region for each location ℓ, an arbitrary subset $\mathsf{Inv}(\ell) \subseteq \mathbb{R}^n$ as evolution domain restriction, arbitrary relation $\mathsf{Flow}(\ell) \subseteq \mathbb{R}^n \times \mathbb{R}^n$ as flow constraints, arbitrary relation $\mathsf{Jump}(e) \subseteq \mathbb{R}^n \times \mathbb{R}^n$ jump conditions, and an arbitrary subset $\mathsf{Final}(\ell) \subseteq \mathbb{R}^n$ as final conditions. Generalizations like those have been suggested in the literature numerous times. They are useful as mathematical models, but not for any computational or verification purpose. It is important to understand why.

We can only obtain meaningful model checking results for a hybrid automaton if we can describe the hybrid automaton (e.g., as an input file in a computer for the model checker). There is no way to describe arbitrary sets $\mathsf{Init}(\ell), \mathsf{Inv}(\ell) \subseteq \mathbb{R}^n, \mathsf{Jump}(e) \subseteq \mathbb{R}^n \times \mathbb{R}^n$ etc. as inputs, because there are uncountable many such sets, but model checkers accept only finite input files from a countable set of inputs.

Moreover, even for cases where there is some description of those sets, we still need to equip the model checker with a way to decide membership in those sets. Suppose some model checking algorithm worked hard to find out that the hybrid automaton will be unsafe when started in a particular state $v \in \mathbb{R}^n$. Then, the model checker still needs to find out whether the hybrid automaton allows v as an initial state or not. That is, we need to give the model checker a way of deciding whether $v \in \mathsf{Init}(\ell)$ for any location $\ell \in \mathsf{Loc}$. Mathematically, this is a simple set inclusion and looks trivial. But that does not mean there would be a computer program that can decide whether $v \in \mathsf{Init}(\ell)$ or $v \notin \mathsf{Init}(\ell)$. For arbitrary sets $\mathsf{Init}(\ell) \subseteq \mathbb{R}^n$, this is impossible by classical results on the limits of computation due to Turing, Church, Gödel, and others. The Mandelbrot set is an example of such a set $\mathsf{Init}(\ell)$ for which it is impossible to decide membership even in a very strong model of real computation [32].

Similar observations hold for all the other parts of hybrid automata. Consequently, we have to assume more structure on $\mathsf{Init}(\ell)$ and all the parts of the definitions of hybrid automata. This is the reason why it is crucial that Definition 6 requires hybrid automata to be described in a definable way. Definition 6 requires hybrid automata to be described by polynomial constraints with rational coefficients, which are representable on a computer, unlike constraints with any arbitrary real coefficients. This also explains why it is critical to restrict polyhedra to rational coefficients in Definition 5.

It should be noted that these observations about the requirements on hybrid automata are crucial for all model checkers, whether they try to decide fragments or semidecide fragments or whether they just strive to approximately answer the reachability problem. Fundamental limits of computation that represent the *numerical analogue of the halting problem* otherwise cause strong undecidabilities even for approximate answers [147], unless additional assumptions are imposed on the hybrid automata [51, 147].

4 Set-based Reachability Analysis

There are two kinds of events that can take place in a hybrid automaton: time can pass with the state evolving according to the flow constraints, or a jump can take the system instantaneously to a new state. Starting from the initial states, *set-based reachability analysis* exhaustively computes the successor states for both time elapse and jumps in alternation until this no longer produces any new states. Since this process might not terminate (see decidability results in Sect. 3), an a-priori limit on the search depth is sometimes imposed. The search depth is usually counted in the number of jumps and, in analogy to discrete automata, this is referred to as *bounded model checking*.

Reachability computation can be seen as a generalization of *numerical simulation*. In numerical simulation, one picks an initial state and tries to compute a successor state that lies on one of the solutions of the corresponding flow constraint and also satisfies one of the jump conditions (some intermediate points along the trajectory are usually kept as well). Then one picks one of the successor states of the jump and repeats the process. Like numerical simulation, reachability analysis directly follows the transition semantics of hybrid automata (Definition 7), but considers sets of states instead of single states.

Just like numerical simulation, reachability computation has to use approximations if the dynamics of the system are complex. Working with sets instead of points, approximate reachability can be conservative in the sense that the computed sets are sure to cover all solutions. Computation costs generally increase sharply in terms of the number of continuous variables. Scalable approximations are available for certain types of dynamics, as discussed later in this section, but this performance comes at a price in accuracy. The trade-off between runtime and accuracy remains a central problem in reachability analysis. Surveys of reachability techniques for hybrid automata can be found, e.g., in [119, 7, 117, 165].

4.1 Reachability Algorithm

The standard method to compute the reachable states is to iterate the following *one-step successor* operators for discrete and continuous transitions. Given a set of states S, let $\mathsf{Post}_C(S)$ be the set of states reachable by letting time elapse from any of the states in S,

$$\mathsf{Post}_C(S) = \{(\ell, w) \mid \exists (\ell, v) \in S : (\ell, v) \xrightarrow{\mathsf{time}} (\ell, w)\}.$$

Let $\mathsf{Post}_D(S)$ be the set of states resulting from taking a discrete transition from any of the states in S,

$$\mathsf{Post}_D(S) = \{(k, w) \mid \exists (\ell, v) \in S, \exists \sigma \in \mathsf{Lab} : (\ell, v) \xrightarrow{\sigma} (k, w)\}.$$

The reachable states are obtained by applying $\mathsf{Post}_C(S)$ and $\mathsf{Post}_D(S)$ in alternation and recording all states that are obtained. The basic algorithm for *forward reachability* computes the following sequence, starting from the initial states:

$$\begin{aligned} R_0 &= \{(\ell,v) \mid v \in [\![\mathsf{Init}(\ell)]\!]\}, \\ R_{i+1} &= R_i \cup \mathsf{Post}_C(R_i) \cup \mathsf{Post}_D(R_i) \qquad \text{for } i = 0,1,2,\ldots. \end{aligned}$$

The algorithm terminates when a fixed-point is reached, i.e., when $R_{i+1} = R_i$ for some $i \geq 0$ (note that $R_i \subseteq R_{i+1}$ for all $i \geq 0$). This simple algorithm does not necessarily terminate, even for systems where reachability is decidable. E.g., a system with an (unbounded) counter would enter a new state at each iteration such that the fixed-point is never reached. Abstraction techniques such as *widening* [86, 22] are used in program analysis to ensure termination, and while they have been applied to hybrid systems with simple dynamics [92] it is difficult to obtain finite-state abtractions for more general cases.

Reachability with symbolic states. A semi-algorithm used frequently for reachability of hybrid automata is shown as Algorithm 1. The states of the hybrid automaton H are represented by finite sets of *symbolic states* (ℓ,P), where $\ell \in \mathsf{Loc}$ and P is a set of continuous states in a suitable set representation such as polyhedra. The set of states corresponding to such a set $R = \{(\ell_1,P_1),(\ell_2,P_2),\ldots\}$ is

$$[\![R]\!] = \{(\ell,v) \mid \exists(\ell,P) \in R : v \in P\}.$$

If H is safe, Algorithm 1 computes the reachable states by iterating one-step successor computations on such a set R, without guarantee of termination. If H is not safe, the procedure will eventually stop when a nonempty intersection of R with the final states is found. A similar semi-algorithm implements the backward approach by iterating a one-step predecessor operator. Other approaches are possible such as mixed forward-backward, where the forward and backward algorithms are executed in an interleaved fashion [92]. All those variations are semi-algorithms since the problem is undecidable.

The one-step successors $\mathsf{Post}_C(S)$ and $\mathsf{Post}_D(S)$ are implemented for symbolic states by enumerating over locations and transitions, respectively, using the following operators. The *continuous successors* of a set of continuous states P in a location ℓ is the set of continuous states

$$\mathsf{post}_\ell(P) = \{x' \mid \exists x \in P : (\ell,x) \xrightarrow{\mathsf{time}} (\ell,x')\}.$$

Similarly, the *discrete successors* of set of continuous states P for an edge $\varepsilon = (\ell,\sigma,k)$ is the set of continuous states

$$\mathsf{post}_\varepsilon(P) = \{x' \mid \exists x \in P : (\ell,x) \xrightarrow{\sigma} (k,x')\}.$$

Formally, the one-step successors of a set of symbolic states R are expressed using the above operators as

Algorithm 1: A reachability semi-algorithm using symbolic states

```
Input   : A hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$.
Output  : If $H$ is safe then SAFE else UNSAFE.
begin
    $Bad \leftarrow \{(\ell, [\![\mathsf{Final}(\ell)]\!]) \mid \ell \in \mathsf{Loc}\}$ ;
    $R \leftarrow \{(\ell, \mathsf{post}_\ell([\![\mathsf{Init}(\ell)]\!])) \mid \ell \in \mathsf{Loc}\}$ ;
    $R_{old} \leftarrow \varnothing$ ;
    while $[\![R]\!] \not\subseteq [\![R_{old}]\!]$ do
        $R_{old} \leftarrow R$ ;
        $R \leftarrow \{(\ell, \mathsf{post}_\ell(\mathsf{post}_\varepsilon(P))) \mid (\ell, P) \in R \wedge \varepsilon = (\ell, \sigma, k) \in \mathsf{Edg}\}$ ;
        if $[\![R]\!] \cap [\![Bad]\!] \neq \varnothing$ then return UNSAFE;
    return SAFE ;
end
```

$$\mathsf{Post}_C([\![R]\!]) = [\![\{(\ell, \mathsf{post}_\ell(P)) \mid \exists (\ell, P) \in R\}]\!],$$

$$\mathsf{Post}_D([\![R]\!]) = [\![\{(k, \mathsf{post}_\varepsilon(P)) \mid \exists (\ell, P) \in R, \varepsilon = (\ell, \sigma, k) \in \mathsf{Edg}\}]\!].$$

In the following, we discuss the above successor operators $post_\ell(P)$, $\mathsf{post}_\varepsilon(P)$ for different classes of hybrid automata with increasingly complex continuous dynamics. We will focus mainly on computing time elapse successors, since this operation usually dominates costs. Other operations of the reachability algorithm may also become bottlenecks, e.g., computing the discrete successors, containment checking, and clustering.

4.2 Piecewise Constant Dynamics

Hybrid automata with piecewise constant dynamics (PCDA) are a special case of hybrid automata with polynomial dynamics (Definition 6), where all constraints are conjunctive linear and the flow constraints are linear predicates over dotted variables only. That is, the derivatives of the variables are independent of the current continuous state. They are also called *linear hybrid automata* (LHA), where the term linear refers to trajectories instead of dynamics (they do not allow the linear dynamics discussed in the next section). In order to avoid possible confusion resulting from this terminology, we prefer the name PCDA.

Definition 9 (Hybrid automaton with piecewise constant dynamics). A hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$ with polynomial dynamics is called *hybrid automaton with piecewise constant dynamics* iff:

- Init, Inv, Final are conjunctive linear constraints over X,
- Flow are conjunctive linear constraints over $\dot{X}$, and
- Jump are conjunctive linear constraints over $X \cup X^+$.

PCDA are of particular interest to formal verification because the one-step successors can be computed exactly, which is not the case for the more complex dynamics discussed in later sections.

Examples for flow constraints of a PCDA include differential inclusions such as $\dot{x} \in [1,2]$, and conservation laws such as $\dot{x}+\dot{y}=0$. The jump constraints of a PCDA admit arbitrary linear updates of the variables, which can generate complex behavior. For example, PCDA can model discrete-time affine systems, a widely used class of control systems, by using jump constraints of the form $x^{+}=Ax+b$. Chaotic behavior can arise in PCDA due to switching flows [42] or guarded jumps, with which one can model piecewise affine maps such as the tent map [48].

Continuous successors. In the following, we discuss computing the states reachable by time elapse in a given location ℓ of a PCDA and write x as shorthand for the state (ℓ,x). By definition, a trajectory can be an arbitrarily curved function as long as it is differentiable and satisfies both flow constraint and evolution domain restriction. For the purposes of reachability, it suffices to consider only straight-line trajectories of PCDA, as formalized in the following lemma.

Lemma 1. *[13] In any given location of a PCDA, there is a trajectory $\xi(t)$ from $x=\xi(0)$ to $x'=\xi(r)$ for some $r>0$ iff $\eta(t)=x+qt$ with $q=\frac{x'-x}{r}$ is a trajectory from x to x'.*

Using this lemma, we now show that the states reachable by time elapse from a polyhedral set of states P are given by the union of P with a polyhedron that is readily computable [8, 28]. Consider polyhedra P and Q. The states on straight line trajectories starting in P with constant derivative $\dot{x}=q$ for any $q \in Q$ are the *time successors*

$$P \nearrow Q = \{x' \mid x \in P, q \in Q, t \in \mathbb{R}^{\geq 0}, x' = x+qt\}. \tag{1}$$

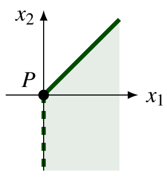
We now transform the right-hand term of (1) into a linear constraint. Let P and Q be polyhedra given in vector-matrix form as $P=\{x \mid Ax \bowtie b\}$, $Q=\{q \mid \bar{A}q \,\bar{\bowtie}\, \bar{b}\}$. By separating the case $t=0$ from $t>0$ in (1) we have $q=\frac{x'-x}{t}$. Eliminating q and multiplying with t yields

$$P \nearrow Q = P \cup \left\{x' \;\middle|\; Ax \bowtie b \,\wedge\, \bar{A}(x'-x) \,\bar{\bowtie}\, \bar{b}\cdot t \,\wedge\, t>0\right\}. \tag{2}$$

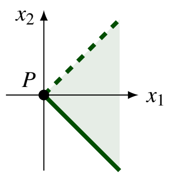
The right-hand term of the union in (2) is a polyhedron that can be computed by quantifier elimination over $X \cup \{t\}$ using, e.g., Fourier-Motzkin elimination. If Q is closed and bounded, the constraint $t>0$ in (2) can be replaced by $t \geq 0$, so the right-hand term contains P and $P \nearrow Q$ becomes a single polyhedron. The following example illustrates that $P \nearrow Q$ can be the union of two polyhedra.

Example 1. For $P=\{x_1=0 \wedge x_2=0\}$, and Q, Q' given in Fig. 3, (2) yields

$$\begin{aligned}P \nearrow Q &= P \cup \{(x_1',x_2') \mid x_1=0 \wedge x_2=0 \wedge x_1'-x_1=t \wedge x_2'-x_2 \leq t \wedge t>0\}\\ &= P \cup \{(x_1',x_2') \mid x_1'=t \wedge x_2' \leq t \wedge t>0\} = P \cup \{(x_1',x_2') \mid x_1'>0 \wedge x_2' \leq x_1'\}.\end{aligned}$$

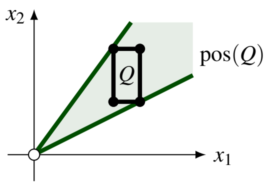


(a) $P \nearrow Q$ with closed and unbounded $Q = \{q_1 = 1 \wedge q_2 \leq 1\}$

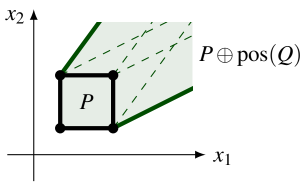


(b) $P \nearrow Q'$ with bounded and non-closed $Q' = \{q_1 = 1 \wedge -1 \leq q_2 < 1\}$

Fig. 3 Given a polyhedron P and a polyhedral set of derivatives Q, the time successors $P \nearrow Q$ can be a convex set that is not a single polyhedron but the union of P with another polyhedron



(a) Q and its cone pos(Q)



(b) P and $P \nearrow Q = P \cup P \oplus \text{pos}(Q)$

Fig. 4 The time successors $P \nearrow Q$ using geometric operations on polyhedra P and Q

Here, the closed but unbounded set Q results in a convex set $P \nearrow Q$ that is not a polyhedron but the union of two polyhedra. Similarly, the bounded but non-closed set Q' results in $P \nearrow Q' = P \cup \{x'_1 > 0 \wedge -x'_1 \leq x'_2 < x'_1\}$, which is also convex and not a polyhedron.

The time successor operation can also be carried out using geometrical operations on the polyhedra P and Q as shown in Fig. 4 [86]. The *positive cone* of Q is the polyhedral set $\text{pos}(Q) = \{q \cdot t \mid q \in Q, t > 0\}$. The time successors are given by the Minkowski sum[3] with P and the positive cone of Q,

$$P \nearrow Q = P \cup (P \oplus \text{pos}(Q)) \tag{3}$$

If P and Q are closed with generator representation (V,R) and (V',R'), respectively, then a generator representation of $P \nearrow Q$ is $(V, R \cup V' \cup R')$.

It remains to ensure that the time successors are reachable by trajectories that satisfy $\mathsf{Inv}(\ell)$. Assuming that $P \subseteq [\![\mathsf{Inv}(\ell)]\!]$, this restriction reduces to $x' \in [\![\mathsf{Inv}(\ell)]\!]$ since $[\![\mathsf{Inv}(\ell)]\!]$ is convex and only straight line trajectories need to be considered. This leads us to the following discrete successor operator for PCDA.

Lemma 2. *[8] The* continuous successors *of a polyhedron P in a location ℓ of a PCDA H is the set:*

$$\mathsf{post}_\ell(P) = \big(P \nearrow [\![\mathsf{Flow}(\ell)]\!]\big) \cap [\![\mathsf{Inv}(\ell)]\!].$$

[3] The *Minkowski sum* is defined as $P \oplus Q = \{p + q \mid p \in P, q \in Q\}$.

Discrete successors. The *discrete successors* of a polyhedron P for an edge $\varepsilon = (\ell, \sigma, k)$ of a PCDA H is the set:

$$\mathsf{post}_\varepsilon(P) = \left\{x^+ \;\middle|\; \exists x \in P : (x,x^+) \in [\![\mathsf{Jump}(\varepsilon)]\!] \wedge x^+ \in [\![\mathsf{Inv}(k)]\!]\right\}.$$

This set is defined using existential quantification, and computing it may require costly quantifier elimination. Frequently occurring special cases can be computed more efficiently. As an example, consider $\mathsf{Jump}(e)$ given by a guard $x \in G$ and a reset $x^+ = Cx + d$, with a constant matrix C and a vector d of appropriate dimensions. The discrete successors are

$$\mathsf{post}_\varepsilon(P) = \big(C(P \cap G) \oplus \{d\}\big) \cap [\![\mathsf{Inv}(k)]\!]. \tag{4}$$

If C is invertible and all sets are polyhedra in constraint representation, the computation is straightforward since intersection corresponds to concatenation of constraints, and for any polyhedra $Q = \{x \mid Ax \bowtie b\}$,

$$CQ \oplus \{d\} = \{x \mid AC^{-1}x \bowtie b + C^{-1}d\}.$$

Computational cost. Computing the continuous successors using (3) involves the cone, Minkowksi sum and intersection operations, for details see [86, 22]. The cone and Minkowski sum are efficient only in the generator representation of a polyhedron (see Definition 5). The intersection operation is efficient only in constraint representation. Translating the polyhedron from constraints to generators and vice versa can produce a number of generators that is exponential in the number of variables. E.g., consider that a n-dimensional cube has $2n$ constraints and 2^n vertices. Dually, a n-dimensional *cross-polytope* (hyperoctahedron) has $2n$ vertices and 2^n constraints. In total, the cost of computing the continuous successors is exponential in the number of variables. Tools such as HyTech and PHAVer use the geometric version (3) of the time successor operator since in practice it is often more efficient than quantifier elimination [87]. The operator is available in computational geometry libraries such as the Parma Polyhedra Library (PPL) [22].

The cost of computing the discrete successors is exponential for polyhedra in constraint representation since it involves quantifier elimination. For some frequently occurring special cases the cost is polynomial, e.g., in the case of (4) with invertible map.

The containment and emptiness tests in Algorithm 1 are carried out pairwise over the elements of sets of symbolic states. The *containment test* $P \subseteq Q$ is solvable with linear programming (and thus in polynomial time) if P, Q are in constraint representation.[4] The *emptiness test* $P = \varnothing$ is solvable as a linear program if P is in constraint representation and trivial if P is in generator representation.

[4] Checking $P \subseteq Q$ is polynomial unless P is in constraint representation and Q is in generator representation, in which case it is known to be NP-complete [71].

Path constraints. A *path* of a hybrid automaton is a sequence of adjacent edges (usually from an initial to a final location). An interesting property of PCDA is that the reachable states along a given path can be encoded by a conjunction of linear constraints, the so-called *path constraints*. The reachability problem for a given path can therefore be solved very efficiently using linear programming. This approach has been implemented in the tool BACH [37]. The number of paths in a PCDA can be infinite if there are cycles, so techniques such as CEGAR have been used to reduce the number of paths to be checked and accelerate termination [162].

4.3 Piecewise Affine Dynamics

Hybrid automata with piecewise affine dynamics (PWA) are a special case of hybrid automata with polynomial dynamics (Definition 6), where all constraints are linear and the flow constraints are linear ordinary differential equations (ODEs). We divide the continuous variables into *state variables* $X = \{x_1, \dots, x_n\}$, whose derivative is explicitly defined, and *input variables* $U = \{u_1, \dots, u_m\}$, whose derivative is unconstrained. The input variables can be used to model nondeterminism such as open inputs to the system, approximation errors, disturbances, etc.

In each location of a PWA, the continuous dynamics are *affine*, i.e., given by differential equations of the form

$$\dot{x} = Ax + Bu, \qquad u \in \mathcal{U}, \tag{5}$$

where A and B are matrices of appropriate dimension and the *input set* $\mathcal{U}$ is compact and convex. Note that some differential inclusions can be brought to this form by introducing auxiliary variables. Similarly, jump constraints of an edge e define resets of the form

$$x^+ = Cx + Du, \tag{6}$$

where x^+ denotes the value of x after the jump, u is defined as above and C and D are matrices of appropriate dimension. The jump constraints also define a set $\mathcal{G}$ called the *guard* of the edge, and a jump can only take place of $x \in \mathcal{G}$. The formal definition of PWA is as follows.

Definition 10 (Hybrid automaton with piecewise affine dynamics). A *hybrid automaton with piecewise affine dynamics* is a hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X \cup U, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$ where

- Init and Inv are conjunctive linear constraints over X.
- Inv are conjunctive linear constraints over $X \cup U$, such that each linear term ranges over variables exclusively from either X or U (no correlation between state and input variables). The input set $\mathcal{U}$ of a location ℓ is given by the terms of $\mathsf{Inv}(\ell)$ that range over U and must be closed and bounded.
- Flow are constraints over $\dot{X} \cup X \cup U$ of the form $\dot{x} = Ax + Bu$.

- Jump are conjunctive linear constraints over $X^+ \cup X \cup U$ whose terms range either over X or are of the form $x^+ = Cx + Du$. The *guard* set $\mathcal{G}$ of an edge e is given by the terms of $\mathsf{Jump}(e)$ that range over X.

The reachable states of a PWA can be computed using Algorithm 1 from Sect. 4.2, with suitable operators post_ℓ for continuous and post_e for discrete successors that will be presented in the following section.

4.3.1 Successor Computations

The successor computations for affine dynamics can be approximated by sequences of geometric set operations. We first present such a sequence for the continuous successors, then give the equation for the discrete successors. Different set representations can be used to implement these operations, and a selection is discussed in the subsequent Sect. 4.3.2.

Continuous successors. In the following, we discuss how to compute the states reachable by time elapse in a given location ℓ. Since ℓ is clear from the context we call x a (continuous) state. We will initially ignore any evolution domain restriction on x and discuss it after the basic construction has been presented. The evolution of the input variables is described by an *input signal* $\zeta : \mathbb{R}^{\geq 0} \to \mathcal{U}$ that attributes to each point in time a value of the input u. The input signal does not need to be continuous. A trajectory $\xi(t)$ from a state x_0 is the solution of the differential equation (5) for initial condition $\xi(0) = x_0$ and a given input signal ζ. It has the form

$$\xi_{x_0,\zeta}(t) = e^{At}x_0 + \int_0^t e^{A(t-s)}B\zeta(s)ds. \tag{7}$$

It consists of the superposition of the solution of the *autonomous* system, obtained for $\zeta(t) = 0$, and the input integral obtained for $x_0 = 0$. In the following, this decomposition of (7) will be exploited to obtain efficient and accurate approximations. A state x' is reachable from some initial set of states $\mathcal{X}_0$ in time t if for some $x_0 \in \mathcal{X}_0$ and some ζ, $x' = \xi_{x_0,\zeta}(t)$. We now describe the reachable states as sets using (7). Let $\mathcal{X}_t$ be the states reachable in time t from any state in $\mathcal{X}_0$ and let $\mathcal{Y}_t$ be the states reachable from $\mathcal{X}_0 = \{0\}$, then (7) can be written as

$$\mathcal{X}_t = e^{At}\mathcal{X}_0 \oplus \mathcal{Y}_t. \tag{8}$$

The goal is to conservatively approximate the reachable states over some finite time horizon T, i.e., to compute a finite sequence of sets $\Omega_0, \Omega_1, \ldots$ such that

$$\bigcup_{0 \leq t \leq T} \mathcal{X}_t \subseteq \Omega_0 \cup \Omega_1 \cup \ldots. \tag{9}$$

We present the construction of a sequence of Ω_k for a fixed *sampling time* $\delta > 0$ such that Ω_k covers $\mathcal{X}_t$ for $t \in [k\delta, (k+1)\delta]$, as illustrated in Fig. 5. The so-called

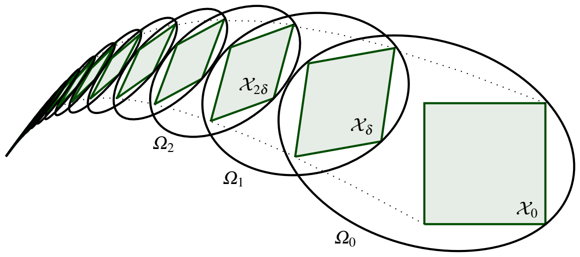


Fig. 5 A sequence of sets $\Omega_0, \Omega_1, \ldots$ that covers $\mathcal{X}_t$ over a finite time horizon T. The choice of set representation for Ω_k (illustrated here by ellipsoids) has a substantial impact on accuracy and computational complexity

semi-group property of reachability says that, starting from $\mathcal{X}_s$, for any $s \geq 0$, and then waiting r time units leads to the same states as starting from $\mathcal{X}_0$ and waiting $r+s$ time units. Applying this to (8), we obtain that for any $r, s \geq 0$,

$$\mathcal{X}_{r+s} = e^{Ar}\mathcal{X}_s \oplus \mathcal{Y}_r. \tag{10}$$

Substituting $r \leftarrow \delta$, $s \leftarrow k\delta$, we obtain a recursive time discretization in the form of

$$\mathcal{X}_{(k+1)\delta} = e^{A\delta}\mathcal{X}_{k\delta} \oplus \mathcal{Y}_\delta.$$

It follows that if we have initial approximations Ω_0 and Ψ_δ such that

$$\bigcup_{0 \leq t \leq \delta} \mathcal{X}_t \subseteq \Omega_0, \qquad \mathcal{Y}_\delta \subseteq \Psi_\delta, \tag{11}$$

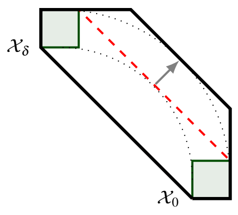
then the sequence

$$\Omega_{k+1} = e^{A\delta}\Omega_k \oplus \Psi_\delta. \tag{12}$$

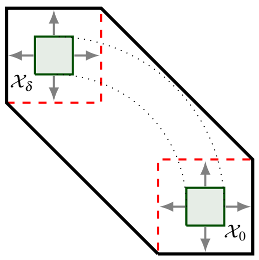
satisfies (9). Note that Ω_0 covers the reachable set over an interval of time $[0, \delta]$, while Ψ_δ covers the values of the input integral at a single time instant δ.

Computing initial approximations Ω_0 and Ψ_δ . The set Ω_0 needs to cover $\mathcal{X}_t$ from $t = 0$ to $t = \delta$. A good starting point for such a cover is the convex hull of $\mathcal{X}_0$ and $\mathcal{X}_\delta$. One approach, shown in Fig. 6(a), is to compute the convex hull in constraint representation, and push the facets out far enough to be conservative [81]. The required values can be computed from a Taylor approximation of (7) [19], or by solving an optimization problem [45]. Note that the cost of computing the exact constraints of the convex hull can be exponential in the number of variables, which limits the scalability of this approach.

A scalable way to obtain Ω_0 is to bloat $\mathcal{X}_0$ and $\mathcal{X}_\delta$ enough to compensate for the curvature of trajectories [75], as illustrated in Fig. 6(b). We present the approach from [75], which uses uniform bloating and whose approximation error is asymptotically linear in the time step δ as $\delta \to 0$. This is asymptotically optimal for any approximation containing the convex hull of $\mathcal{X}_0$ and $\mathcal{X}_\delta$ [110]. The bloating can



(a) pushing facets



(b) bloating

Fig. 6 An approximation Ω_0 that covers $\mathcal{X}_t$ for $t \in [0,\delta]$ can be obtained from the convex hull of $\mathcal{X}_0$ and $\mathcal{X}_\delta$ and enlarging it enough to compensate for the curvature of trajectories

be made non-uniform in space and time to obtain a more precise approximation [110, 68]. The bloating factor is derived from a Taylor approximation of (7), whose remainder is bounded using norms. To formalize the above statements, we use the following notation. Let $\|\cdot\|$ be a vector norm and let $\|A\|$ be its induced matrix norm.[5] Let $\mu(\mathcal{X}) = \max_{x\in\mathcal{X}} \|x\|$ and let $\mathcal{B}$ be the unit *ball* of the norm, i.e., the largest set $\mathcal{B}$ such that $\mu(\mathcal{B}) = 1$. For a scalar c, let $c\mathcal{X} = \{cx \mid x \in \mathcal{X}\}$. The approximation error is measured using the *Haussdorff distance* between sets $\mathcal{X}, \mathcal{Y}$,

$$d_H(\mathcal{X},\mathcal{Y}) = \max\Big\{\sup_{x\in\mathcal{X}} \inf_{y\in\mathcal{Y}} \|x-y\|, \sup_{y\in\mathcal{Y}} \inf_{x\in\mathcal{X}} \|x-y\|\Big\}.$$

Lemma 3. *[75] Given a set of initial states $\mathcal{X}_0$ and affine dynamics (5), let*

$$\begin{aligned}
\alpha_\delta &= \mu(\mathcal{X}_0)\cdot(e^{\|A\|\delta} - 1 - \|A\|\delta),\\
\beta_\delta &= \tfrac{1}{\|A\|}\mu(B\mathcal{U})\cdot(e^{\|A\|\delta} - 1),\\
\Omega_0 &= \operatorname{chull}(\mathcal{X}_0 \cup e^{A\delta}\mathcal{X}_0) \oplus (\alpha_\delta + \beta_\delta)\mathcal{B},\\
\Psi_\delta &= \beta_\delta \mathcal{B}.
\end{aligned}$$

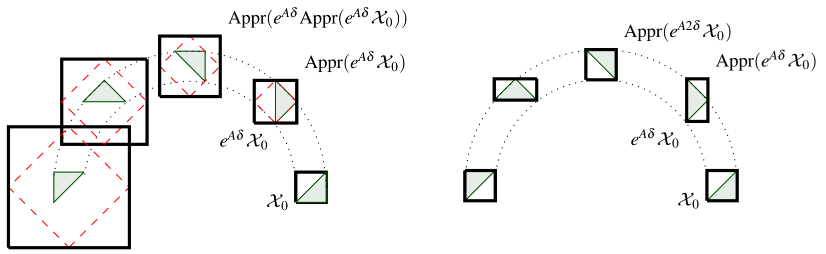
Then $\bigcup_{0\le t\le\delta} \mathcal{X}_t \subseteq \Omega_0$ and $\mathcal{Y}_\delta \subseteq \Psi_\delta$. Furthermore, if $B\mathcal{U}$ is a ball of the norm, i.e., $B\mathcal{U} = \mu(B\mathcal{U})\mathcal{B}$, the approximation error is bounded by

$$\begin{aligned}
d_H\big(\textstyle\bigcup_{0\le t\le\delta} \mathcal{X}_t, \Omega_0\big) &\le \delta e^{\|A\|\delta}\big(\mu(B\mathcal{U}) + (\tfrac{1}{2}+\delta)\|A\|\mu(\mathcal{X}_0)\big),\\
d_H\big(\mathcal{Y}_\delta, \Psi_\delta\big) &\le \delta^2\|A\|e^{\|A\|\delta}\mu(B\mathcal{U}).
\end{aligned}$$

Propagating the initial approximation Ω_0 forward in time using (12) gives an approximation of $\mathcal{X}_t$ over a bounded horizon. The following theorem gives a bound on the total approximation error.

Theorem 3. *[75] Given Ω_0 and Ψ_δ as defined in Lemma 3, let $\Omega_{k+1} = e^{A\delta}\Omega_k \oplus \Psi_\delta$ for $k = 1,\dots,N-1$. Then $\bigcup_{0\le t\le N\delta} \mathcal{X}_t \subseteq \bigcup_{0\le k\le N-1} \Omega_k$. Furthermore, if $B\mathcal{U}$ is a ball*

[5] For example, the *infinity norm* $\|x\|_\infty = \max\{|x_1|,\dots,|x_n|\}$ induces the matrix norm $\|A\| = \max_{1\le i\le n}\sum_{j=1}^{m}|a_{ij}|$, where A is of dimension $n\times m$. Its ball $\mathcal{B}_\infty$ is a cube of side length 2.



$\mathrm{Appr}(e^{A\delta}\mathrm{Appr}(e^{A\delta}\mathcal{X}_0))$
$\mathrm{Appr}(e^{A\delta}\mathcal{X}_0)$
$e^{A\delta}\mathcal{X}_0$
$\mathcal{X}_0$
$\mathrm{Appr}(e^{A2\delta}\mathcal{X}_0)$
$\mathrm{Appr}(e^{A\delta}\mathcal{X}_0)$
$e^{A\delta}\mathcal{X}_0$
$\mathcal{X}_0$

(a) approximation with wrapping effect (b) using a wrapping-free algorithm

Fig. 7 The wrapping effect can lead to an exponential increase in the approximation error that can be avoided for affine dynamics. This example shows the exact solution $e^{Ak\delta}\mathcal{X}_0$ (shaded) and an interval hull approximation (thick), with $e^{A\delta}$ performing a rotation of 45 degrees around the origin. The wrapping effect occurs if the approximation is applied to the map of the previous approximation (dashed). To illustrate the effect more clearly, $\mathcal{X}_0$ is used here instead of Ω_0

of the norm, the approximation error is bounded by

$$d_H\left(\bigcup_{0\le t\le N\delta}\mathcal{X}_t,\bigcup_{0\le k\le N-1}\Omega_k\right) \le \delta e^{\|A\|N\delta}\left(2\mu(B\mathcal{U})+(\tfrac{1}{2}+\delta)\|A\|\mu(\mathcal{X}_0)\right)$$

Approximations and the wrapping effect. The sequence in (12) can be problematic to compute since the complexity of Ω_k may increase sharply with k. We illustrate this for the case where Ω_k is a polytope in generator representation, and a similar argument can be made for constraint representation. Let N_k be the number of vertices of Ω_k and let Ψ_δ have M vertices. Since Ω_{k+1} is the sum of $e^{A\delta}\Omega_k$ with Ψ_δ it can have $N_{k+1} = N_k \cdot M$ vertices. Resolving the recursion, we get the tight upper bound $N_k \le N_0 \cdot M^k$. To avoid this increase in complexity, we approximate each Ω_k by a simplified set. Let Appr be an *approximation function* such that for any set P, $P \subseteq \mathrm{Appr}(P)$. The sequence (12) then becomes

$$\hat{\Omega}_{k+1} = \mathrm{Appr}(e^{A\delta}\hat{\Omega}_k \oplus \Psi_\delta). \tag{13}$$

For example, if Appr computes the interval hull (bounding box) and Ω_0 is a polytope, then all $\hat{\Omega}_k$ are polytopes with $2n$ facets. However, the recursive application of the approximation function can lead to an exponential increase in the approximation error. This phenomenon is known in numerical analysis as the *wrapping effect* [105] and is illustrated in Fig. 7.

For affine dynamics, the wrapping effect can be avoided by combining two techniques [77]. First, the approximation operator is chosen such that it distributes over Minkowski sum, i.e., $\mathrm{Appr}(P \oplus Q) = \mathrm{Appr}(P) \oplus \mathrm{Appr}(Q)$. This is the case, e.g., for the interval hull (bounding box). Second, the alternation of the map $e^{Ak\delta}$ with the Minkowski sum in (12) is avoided by splitting it into two sequences

$$\begin{aligned} \hat{\Psi}_{k+1} &= \text{Appr}(e^{Ak\delta}\Psi_\delta) \oplus \hat{\Psi}_k, \qquad \text{with } \hat{\Psi}_0 = \{0\}, \\ \hat{\Omega}_k &= \text{Appr}(e^{Ak\delta}\Omega_0) \oplus \hat{\Psi}_k. \end{aligned} \tag{14}$$

For sequence (14) it holds that $\hat{\Omega}_k = \text{Appr}(\Omega_k)$, which means the resulting approximation is free of the wrapping effect. The total approximation error consists of the bounds of Theorem 3 plus the error introduced by the operator Appr (measured in terms of the Hausdorff distance).

The approach is easily extended to variable time steps by adapting Ω_0 and Ψ_δ to the time step while computing the sequence [68].

Evolution domain restriction. So far we have neglected the evolution domain restriction (invariant) $\mathsf{Inv}(\ell)$ of the location. Let $\mathcal{S} = [\![\mathsf{Inv}(\ell)]\!]$. A simple but efficient heuristic tries to find, if it exists, the smallest K such that Ω_K lies completely outside $\mathcal{S}$. The search of such a K may be combined with finding a suitable time horizon T and a suitable time step δ (this search obviously might not terminate). Then one computes the sequence $\Omega_0, \ldots, \Omega_K$ and obtains the sequence $\bar{\Omega}_k = \Omega_k \cap \mathcal{S}$ as an approximation of the continuous successors over the time horizon $T = K\delta$.

In cases where the above solution is overly conservative, one can improve the approximation using the following approach from [83]. Let $\mathcal{S}_t$ be the states reachable from $\mathcal{S}$ (neglecting the evolution domain restriction), and let $\xi(\tau)$ be a trajectory inside $\mathcal{S}$ for all $0 \le \tau \le t$. Then the semi-group property implies that $\xi(\tau + s) \in \mathcal{S}_s$ for all $0 \le s \le t - \tau$, so that $\xi(t) \in \bigcap_{0 \le \tau \le t} \mathcal{S}_\tau$. We may therefore improve the approximation by intersecting Ω_k with an approximation of the states reachable from $\mathcal{S}$, which we obtain from the sequence in (14) with $\Omega_0 \leftarrow \mathcal{S}$. This leads to the following sequence $\bar{\Omega}_k$ that approximates the continuous successors, starting with $k = 0$ and $\Psi_0 = \{0\}$:

$$\begin{aligned} \Psi_{k+1} &= \text{Appr}\big(e^{Ak\delta}\Psi_\delta\big) \oplus \Psi_k, \\ \bar{\Omega}_k &= \big(\text{Appr}(e^{Ak\delta}\Omega_0) \oplus \Psi_k\big) \,\cap\, \textstyle\bigcap_{0 \le i \le k} \big(\text{Appr}(e^{Ai\delta}\mathcal{S}) \oplus \Psi_i\big). \end{aligned} \tag{15}$$

Discrete successors. Consider an edge $\varepsilon = (\ell, \sigma, k)$ of a PWA, whose jump constraints define the reset map

$$x^+ = Cx + Du$$

and the guard set $\mathcal{G}$, which only lets states jump where $x \in \mathcal{G}$. Recall that $u \in \mathcal{U}$, where $\mathcal{U}$ is compact, convex and given by constraints in $Inv(\ell)$. Let $\mathcal{S}^+ = [\![\mathsf{Inv}(k)]\!]$ be the evolution domain restriction of the target location. The discrete successors of a set P can be written using geometric operators as

$$\mathsf{post}_\varepsilon(P) = \big(C(P \cap \mathcal{G}) \oplus D\mathcal{U}\big) \cap \mathcal{S}^+.$$

We now turn to representing the individual sets in the sequences Ψ_k and Ω_k, and which approximation operator Appr to use.

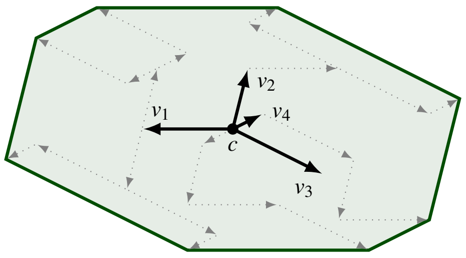


Fig. 8 A zonotope is a special form of centrally symmetric polytope, as illustrated here with generators v_1, v_2, v_3, v_4, and center $c = 0$

4.3.2 Set Representations

Several set representations have been proposed in literature for computing the continuous successors under affine dynamics, using variations of the algorithm presented in the previous section. To be efficient, scalable implementations or approximations need to be available for the operators in the algorithm. Using the initial approximation from Lemma 3 and the recurrence equation (14), the operators are linear map, Minkowski sum, convex hull and intersection. The following paragraphs summarize the results for a selection of prominent representations.

Ellipsoids. The first scalable reachability algorithms for affine dynamics were obtained for ellipsoids, see [133, 107] and references therein. An approximation of the reachable states using ellipsoids is shown in Fig. 5. A nondegenerate *ellipsoid* $\mathcal{E}(c,Q) \subseteq \mathbb{R}^n$ is represented by a center $c \in \mathbb{Q}^n$ and a positive definite[6] matrix $Q \in \mathbb{Q}^{n\times n}$,

$$\mathcal{E}(c,Q) = \left\{x \;\middle|\; (x-c)^\mathsf{T} Q^{-1}(x-c) \leq 1\right\}$$

(this can be generalized to degenerate ellipsoids). Deterministic affine transforms can be computed efficiently for ellipsoids. For a matrix $A \in \mathbb{Q}^{n\times n}$ and vector $b \in \mathbb{Q}^n$,

$$A\mathcal{E}(c,Q) + b = \mathcal{E}(Ac+b, AQA^\mathsf{T}).$$

Ellipsoids are not closed under Minkowski sum, convex hull, nor intersection. Using ellipsoids one therefore generally suffers from the wrapping effect unless $B\mathcal{U}$ is a singleton. Efficient approximations are available for Minkowski sum, convex hull, and special cases of intersection, but the computation of discrete successors can be problematic in terms of accuracy. For an implementation, see [106].

Zonotopes. Zonotopes are a compact representation for a special form of polytopes that have been used successfully for reachability analysis due to their computationally attractive features [75, 3]. A *zonotope* $P \subseteq \mathbb{R}^n$ is defined by a center $c \in \mathbb{Q}^n$ and a finite number of generators $v_1, \ldots, v_k \in \mathbb{Q}^n$ that span the polytope as bounded linear combinations from the center:

[6] A matrix Q is positive definite iff it is symmetric and $x^\mathsf{T} Q x > 0$ for all $x \neq 0$.

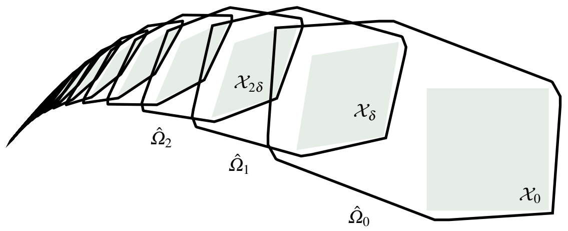


Fig. 9 A reach set cover $\hat{\Omega}_0, \hat{\Omega}_1, \ldots$ (dotted) computed with zonotopes using the implementation in [3] (solid).

$$P = \left\{ c + \sum_{i=1}^{k} \alpha_i v_i \;\middle|\; \alpha_i \in [-1,1] \right\}$$

A common denotation for this zonotope is $P = (c, \langle v_1, \ldots, v_k \rangle)$. A zonotope with k generators is an affine transformation of a k-dimensional unit hypercube. Zonotopes are central-symmetric convex polytopes, see Fig. 8 for an illustration. Affine transformations can be computed efficiently for zonotopes. For a matrix $A \in \mathbb{Q}^{m \times n}$, the image of the linear transformation can simply be computed component-wise:

$$AP = (Ac, \langle Av_1, \ldots, Av_k \rangle)$$

The Minkowski sum can be computed efficiently for zonotopes $P = (c, \langle v_1, \ldots, v_k \rangle)$ and $Q = (d, \langle w_1, \ldots, w_m \rangle)$ by a single vector addition and a single list concatenation:

$$P \oplus Q = (c + d, \langle v_1, \ldots, v_k, w_1, \ldots, w_m \rangle).$$

Since zonotopes are closed under Minkowski sum, it is straightforward to devise an approximation operator Appr that distributes over Minkowski sum and use the wrapping-free sequence (14). When the list of generators of a zonotope becomes large, one can efficiently compute a smaller list that results in a cover of the original zonotope [75].

Zonotopes are neither closed under convex hull, nor under intersection. Efficient approximations exists, and the accuracy of approximating the convex hull in the above reachabililty algorithm can be improved by taking smaller time steps. However, the lack of accuracy in intersections can make the computation of discrete successors with zonotopes problematic. In special cases it can be advantageous to use an approach called *continuization* to avoid the intersection operation, see [5]. Instead of intersecting a set of states with the guard set and then applying the dynamics of the successor location to the result, the states suspected to intersect with the guard set (by some approximative measure) are subjected to nondeterministic dynamics that overapproximate the dynamics both before and after the jump. The dynamics of the successor location are used once enough time steps have been carried out to be sure the set no longer intersects with the guard set.

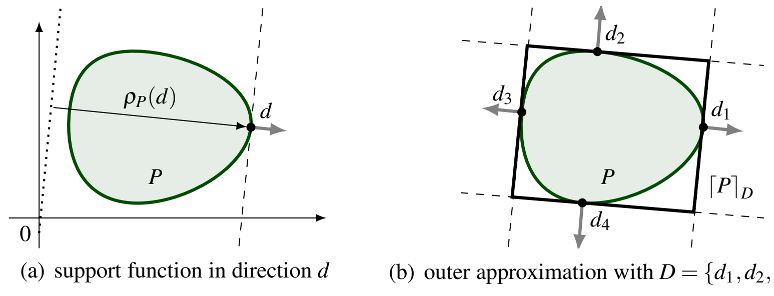

$\rho_P(d)$
d
P
0
(a) support function in direction d
d_2
d_3
d_1
P
$\lceil P \rceil_D$
d_4
(b) outer approximation with $D = \{d_1, d_2, d_3, d_4\}$

Fig. 10 Evaluating the support function in a set of directions gives a polyhedral outer approximations that can be computed very efficiently

Reachability with zonotopes is extremely scalable for affine dynamics [77, 3]. The approach has been extended to nonlinear differential algebraic equations [2].

Support functions. A support function represents a closed, bounded, and convex set exactly, somewhat like a characteristic function. Support functions lead to very scalable algorithms since linear map, Minkowski sum, and convex hull correspond to simple operations on vectors and scalars [74, 116, 83].

The *support function* $\rho_P : \mathbb{R}^n \to \mathbb{R}$ of a nonempty, closed, bounded, and convex set P is

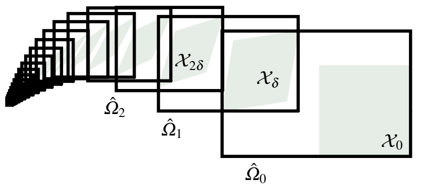
$$\rho_P(d) = \max\{d^\mathsf{T} x \mid x \in P\}.$$

It attributes to every *direction* $d \in \mathbb{R}^n$ the position of the tangent halfspace in that direction, see Fig. 10(a). The values of the support function over a set of directions $D \subseteq \mathbb{R}^n$ define an *outer approximation*

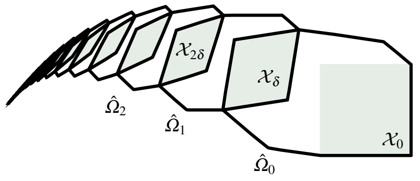
$$\lceil P \rceil_D = \bigcap_{d \in D} \left\{ d^\mathsf{T} x \leq \rho_P(d) \right\}.$$

If $D = \mathbb{R}^n$ or D is the ball of a norm, then $\lceil P \rceil_D = P$, which shows that the support function indeed represents the set exactly. If D is a finite set of directions, the outer approximation is a polyhedron, as illustrated in Fig. 10(b) and applied to reachability in Fig. 11. While for a given direction the numerical value of the support function can often be computed very efficiently, one does not escape the curse of dimensionality if the goal is to compute an outer approximation of a given accuracy: To obtain an outer approximation within a Hausdorff distance ε of P in n dimensions, one needs to evaluate the support function in $\mathcal{O}(1/\varepsilon^{n-1})$ directions. Asymptotically optimal algorithms to construct ε-close approximations are described, e.g., in [116]. However, for some examples even a small number of directions can lead to reachability results with an acceptable approximation error[68].

Linear map, Minkowski sum, and convex hull are easily computed with support functions:



(a) using the axis directions gives a bounding box approximation of Ω_k from (14)



(b) adding more directions, the approximation approaches Ω_k from (14)

Fig. 11 A reach set cover can be computed with support functions and initial approximations Ω_0, Ψ_δ from a variation of Lemma 3 where the bloating is non-uniform [68]. Evaluating the support function in a given set of directions results in the shown outer approximation $\hat{\Omega}_0, \hat{\Omega}_1, \ldots$ (solid)

$$\begin{aligned} \rho_{AP}(d) &= \rho_P(A^\mathsf{T} d), \\ \rho_{P \oplus Q}(d) &= \rho_P(d) + \rho_Q(d), \\ \rho_{\mathrm{chull}(P \cup Q)}(d) &= \max\{\rho_P(d), \rho_Q(d)\}. \end{aligned}$$

The intersection operation is more complex, and can be formulated as an optimization problem [83].

Thanks to the above properties, support functions serve well as a lazy representation for sets that arise from the successor computations described in Sect. 4.3.1. Computing the support function of the sequence (14) for a given direction can be done very efficiently even without the approximation operator Appr [83].

Two issues need to be solved to use support functions efficiently in the reachability Algorithm 1. First, the nesting of support functions should be of limited depth, in particular because evaluating the support function of an intersection operation requires multiple evaluations of its operands. Secondly, deciding containment is hard for support functions. Both problems can be solved by switching the set representation from a support function to its polyhedral outer approximation at appropriate points in the algorithm [68]. Combining support functions and polyhedral computations for a fixed set of directions D is closely related to reachability with *template polyhedra* [160] and both require that a good set of directions D be chosen. The support function representation can be extended to represent the entire (nonconvex) reachable set by parameterizing it over time [69].

Polyhedra. The class of polyhedra is closed under all required operations, i.e., linear map, Minkowski sum, convex hull, and intersection. However, not all of them scale well. As mentioned in Sect. 4.2, there are no scalable algorithms for computing convex hull and Minkowski sum on polyhedra in constraint representation. For illustration, consider that using the convex hull of n line segments, each given by $2n$ constraints in n dimensions, on can construct a cross-polytope, which has 2^n constraints. Taking the Minkowski sum can lead to a similar explosion in the number of constraints. This is illustrated by the fact that the Minkowski sum can be computed with a convex hull and an intersection operation in $n+1$ dimensions using the *Cayley Trick* [173]. A polyhedral approximation for the non-scalable operations can be efficiently computed by a-priori fixing the facet normals of the result, e.g., using the outer approximation of the support function. The accuracy of the approximation can be increased by including additional directions, leading to a scalable approach [20].

4.3.3 Clustering

The accuracy of the approximation in Lemma 3 depends on the size of the time step. This property, common to all approaches cited in Sect. 4.3, points to a potential bottleneck: To achieve a desired accuracy, one may end up with a large number of sets to cover the required time horizon. In the next successor computation, each one of these sets may become the initial set of yet another sequence, and so one may easily end up with an exponential increase in the number of sets. If only very few of these sets intersect with the guard sets, the discrete successor computation results in few sets and therefore acts as a filter that might just keep the number of sets manageable. But this is not the case in general; note that these sets necessarily overlap. To prevent an explosion in the number of sets, a common approach is to cluster together all sets that intersect with the same guard [83]. The clustering operation, e.g., taking the convex hull, can itself be costly and adds to the approximation error in a way that is not easy to quantify. An approach to obtain a suboptimal number of clusters for a given error bound is presented in [69].

4.4 Nonlinear Dynamics

We give a very brief overview of techniques that deal with nonlinear dynamics

$$\dot{x} = f(x),$$

where f is usually assumed to be globally Lipschitz continuous.

Linearization. One way to deal with nonlinear dynamics is to approximate them with affine dynamics $\dot{x} = Ax + u, u \in \mathcal{U}$ and then use reachability algorithms for affine dynamics. First, the states are confined to a bounded domain $\mathcal{S}$. This could be the evolution domain restriction in a location, or $\mathcal{S}$ can be derived iteratively by growing suitable bounds around a given set of initial states. Then, a suitable matrix A and vector b are chosen. For example, linearizing $f(x)$ around a point $x_0 \in \mathcal{S}$ gives a matrix A with elements $a_{ij} = \frac{\partial f_i}{\partial x_j}\big|_{x=x_0}$ and a vector $b = f(x_0) - Ax_0$. Finallly, one derives a set $\mathcal{U}_\varepsilon$ that bounds the error such that for all $x \in \mathcal{S}$,

$$f(x) - (Ax + b) \in \mathcal{U}_\varepsilon.$$

Such bounds can be obtained using, e.g., interval arithmetic or optimization techniques. The states reachable using the affine dynamics $\dot{x} = Ax + u,\ u \in \mathcal{U}_\varepsilon \oplus \{b\}$ cover those of the original nonlinear dynamics. This is approach constructs an *abstraction* of the system. Such abstractions are discussed more formally in Sect. 5.2.

The accuracy of the linearization depends on the size of the domain $\mathcal{S}$. It can be increased by partitioning $\mathcal{S}$ into smaller parts. Each part can then be associated with smaller error bounds $\mathcal{U}_\varepsilon$ and consequently gives a more accurate approximation of the reachable set. The switching of the system from one element of the partition to another is straightforward to model with a hybrid automaton. This process is known as *phase portrait approximation*, see also Sect. 5.2. It can be of use even when dealing with purely continuous dynamical systems, in which case it is also referred to as *hybridization* [18]. The abstract model can be simplified by projecting away variables and adding a clock variable to preserve timing properties [17].

Polynomial Approximations. If the dynamics are polynomial, bringing them to Bernstein form allows one to compute conservative approximations of successors sets in polynomial form [54, 152]. Another approach is to use *Taylor models*, which are polynomial approximations of a functions that are derived from a higher-order Taylor expansion and an interval bound on the remainder [30]. The resulting ODE can be solved by iterative approximations using the Picard operator. The reachable states are approximated by sets that are polyhedra [160] or polynomial images of intervals [43]. A similar approach uses polynomial images of zonotopes, which are themselves images of intervals [4]. Since polynomial images of intervals are generally not closed under intersection, the accuracy may be diminished when computing discrete successors. It can also be shown that additional assumptions, such as knowledge of a Lipschitz constant, are required in these approaches in order to ensure computable error bounds [147].

5 Abstraction-based Verification

Explicit-state reachability analysis is very easy to use. Its flat and direct representation of the system behavior can, however, cause it to run into scalability issues for bigger systems. One technique that has been very successful for scaling up discrete model checking is that of abstraction (see also Chap. **??**).

The basic idea is to replace the actual system by a simpler, abstract system, in which model checking is easier to perform. The verification results about the abstract system, of course, can only be related back to verification results about the original concrete system under certain conditions on how the abstract and concrete system are related and whether the particular property in question survives this abstraction process.

The options for directly constructing discrete abstractions by finite quotients and for which subclasses they work have been examined by Henzinger [88, 93] and Lafferriere et al. [108]. Because of the limited scope of discrete abstractions, more general predicate abstractions [10, 9] and abstraction-refinement techniques like Counterexample-Guided Abstraction-Refinement (CEGAR) have been developed subsequently [46, 9]; see Chap. **??**. Those directions have again worked successfully in discrete and, to some extent, real-time systems.

5.1 Discrete Abstractions

We present a general notion of abstraction for transition systems based on simulation relations [125] and we illustrate the principle of using abstractions in the verification of hybrid systems for the class of initialized rectangular automata.

Definition 11 (Abstraction). A transition system $T^A = \langle S^A, S_0^A, S_f^A, \Sigma, \rightarrow_A \rangle$ is an *abstraction* of a transition system (with same alphabet) $T = \langle S, S_0, S_f, \Sigma, \rightarrow \rangle$ (which is then called the *concrete* system) if there exists an *abstraction mapping* $\alpha : S \rightarrow S^A$ such that the following conditions hold:

1. $\alpha(s) \in S_0^A$ for all initial states $s \in S_0$;
2. for all $\sigma \in \Sigma$, for all states $s_1, s_2 \in S$, if $s_1 \xrightarrow{\sigma} s_2$, then $\alpha(s_1) \xrightarrow{\sigma}_A \alpha(s_2)$;
3. $\alpha(s) \in S_f^A$ for all final states $s \in S_f$.

The abstraction mapping α is in fact a particular case of a (time-abstract) simulation relation [126]. It may be convenient to allow the abstraction mapping to map a state $s \in S$ to several abstract states $s_A^1, s_A^2, \ldots s_A^k \in S^A$, that is to consider abstraction mappings $\alpha : S \rightarrow 2^{S^A}$ or equivalently to consider an abstraction relation over $S \times S^A$, rather than a function. We take the simpler definition which is sufficient for the purpose of describing the main principles of abstraction for hybrid automata.

The main property of abstractions which is useful for safety verification problem of hybrid automata is that they are conservative. Formally, $\{\alpha(s) \mid s \in \mathsf{Reach}(T)\} \subseteq \mathsf{Reach}(T^A)$, which implies the following.

Lemma 4. *Let T^A be an abstraction of T. If T^A is safe, then T is safe.*

By Lemma 4, if we show (e.g., using algorithmic techniques) that the unsafe states are not reachable in an abstraction of a hybrid system, then we can conclude that the concrete system is safe. Intuitively, this is because abstractions are over-approximations of the original system, and therefore they exhibit (or simulate) all executions of the concrete system, and possibly more. In particular, every path to an unsafe state has a matching path in the abstraction, which is the main argument for proving Lemma 4. The converse of this lemma does not hold simply because abstractions may introduce spurious executions (which have no matching execution in the concrete system) due to over-approximation.

The main purpose of abstraction for hybrid system is to obtain finite-state transition systems which are amenable to model-checking by automatic tools, and give useful conclusion about the original system. Remember that the transition system of hybrid automata have (uncountably) infinite state space, and (uncountably) infinite branching. In the next subsections, we present ideas for practically constructing such abstractions.

Initialized rectangular automata. We illustrate abstractions with an informal argument of why the safety verification problem is decidable for initialized rectangular automata. The idea is that for such hybrid automata H, one can construct a timed automaton A such that A is an abstraction of H, and H is an abstraction of A, thus A is safe if and only if H is safe. Note that in this case the constructed abstraction (the timed automaton A) has infinite state space, but since we know that the safety verification problem for timed automata is decidable, we obtain decidability for initialized rectangular hybrid automata by Lemma 4.

We present the main steps behind this construction. In every location, a variable x with flow constraint $k_1 \leq \dot{x} \leq k_2$ is replaced by two variables x_l and x_u with flow constraint $\dot{x}_l = k_1$ and $\dot{x}_u = k_2$ which track the least and greatest possible value of x respectively. An incoming edge with jump condition $a \leq x^+ \leq b$ (an update) is replaced by $x_l^+ = a \wedge x_u^+ = b$. An edge with jump condition $x \leq b$ (a guard) that occurs in conjunction with $x^+ = x$ is replaced by two copy of the edge, one with the constraint $(x_l \leq b \wedge x_u \geq b \wedge x_u^+ = b)$ and the other with the constraint $x_u \leq b$. More complicated jump conditions (strict inequalities, and conjunction of simple jump conditions) are handled analogously, as well as the constraints in initial, final, and evolution domain conditions (invariants).

After this step, the slope of every variable is a singleton in every location. The next step is to scale the nonzero slope of the variables to 1. To do this, in each location we replace flow constraints $\dot{x} = k$ (when $k \neq 0$) by $\dot{x} = 1$ and divide by k the constants in the guards of outgoing edges, and in the updates of incoming edges. This ensures that the value stored in variable x remains k times smaller the value of x in the original automaton (as long as the flow constraint $\dot{x} = k$ holds). It is therefore important that the rectangular automaton is initialized, as it guarantees that if the constraint $x^+ = x$ occurs in the jump condition of an edge (ℓ, σ, ℓ'), then the slope of variable x is the same in ℓ and in ℓ'. It remains to eliminate variables with slope 0, which can be done easily by storing the lower and upper value of x in the finite control structure of the automaton (these values can change only by discrete jumps).

The technical details of how to deal for instance with strict constraints in jump condition like $(a < x < b)$, or unbounded flow constraints (like $\dot{x} \geq 1$) can be found in [96].

5.2 *Phase-portrait approximation*

Phase-portrait approximations are used as abstractions of hybrid automata with complex flow constraints. We discuss the approach for affine flow constraints, but it applies to flow constraints that are much more general (e.g., given by $\dot{x} = f(x)$ for a continuous function f). Details about the theory and practice of this approach can be found e.g. in [93, 67] and extensions on hybridization [18] and other abstractions [60, 70, 169].

The objective of phase-portrait approximations is to replace complex dynamics by simple rectangular (or sometimes linear) flow constraints on the dotted variables only. For example, the flow constraint $\dot{x} = f(x)$ where $f(x) = x + 1$ in a location with evolution domain (invariant) $0 \leq x \leq 10$ is replaced by $1 \leq \dot{x} \leq 11$, which over-approximates the exact dynamics. In general, bounds on the derivative can be derived from bounds on the variables and computed as optimization problems, where the lower bound should be smaller than $\inf_{v \in [\![\mathsf{Inv}(\ell)]\!]} f(v)$ and symmetrically for the upper bound. Manual or numerical methods can be used as long as the bounds can be proven to hold.

Formally, a *phase-portrait approximation* of a hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$ is a hybrid automaton $H' = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}', \mathsf{Jump}, \mathsf{Final} \rangle$ in which all components in H and H' are identical, except the flow constraint which is such that $[\![\mathsf{Flow}'(\ell)]\!] \supseteq [\![\mathsf{Flow}(\ell)]\!]$ for every location $\ell \in \mathsf{Loc}$.

Lemma 5. *Let H' be a phase-portrait approximation of H. Then $[\![H']\!]$ is an abstraction of $[\![H]\!]$, and if $[\![H']\!]$ is safe, then $[\![H]\!]$ is safe.*

The safety verification problem for phase-portrait approximations can be solved using the algorithms and data-structure presented in Sect. 4 for reachability analysis. Rectangular phase-portrait approximation are relatively simple to obtain because bounds are computed for each variable separately. However, the quality of the approximation may be too coarse to establish safety. If the bad states are reachable in the phase-portrait approximation, it may be due to lack of accuracy. More precise approximations are obtained by splitting the evolution domains. For example, a location with evolution domain $0 \leq x \leq 10$ can be replaced by two locations with respective evolution domain $0 \leq x \leq 5$ and $5 \leq x \leq 10$, over which the approximation of $\dot{x} = x + 1$ is more precise, namely $1 \leq \dot{x} \leq 6$ and $6 \leq \dot{x} \leq 11$ respectively. Fig. 12 shows the states reachable from $x = t = 0$ (assuming $\dot{t} = 1$) in the rectangular phase-portrait approximation before splitting (light gray) and after splitting (dark gray).

In general, splitting consists in replacing a location ℓ by k locations $\ell_1, \ldots, \ell_k$ with same flow constraint as in ℓ, and with evolution domains that cover the evolution domain of ℓ, i.e. such that $[\![\mathsf{Inv}(\ell)]\!] \subseteq \bigcup_{i=1}^{k} [\![\mathsf{Inv}(\ell_i)]\!]$. For each incoming edge (ℓ', σ, ℓ), new edges (ℓ', σ, ℓ_i) $(i = 1, \ldots k)$ are created with same jump condition, and similarly for each outgoing edges. The split Locations $\ell_1, \ldots, \ell_k$ are connected by edges with jump condition $stable(X) = \bigwedge_{x \in X} x' = x$. It can be shown that locations splitting results in hybrid automata that are mutually abstractions of each other, implying that one is safe if and only if the other is safe. By splitting locations, rectangular phase-portrait approximation can be made arbitrarily precise in the following sense. Given a hybrid automaton H and $\varepsilon > 0$, an ε-relaxation of H is a hybrid automaton the same locations and transition structure as in H, and where all predicates ϕ in H are replaced by predicate ϕ' such that $[\![\phi]\!] \subseteq [\![\phi']\!] \subseteq [\![\phi]\!]_\varepsilon$ where

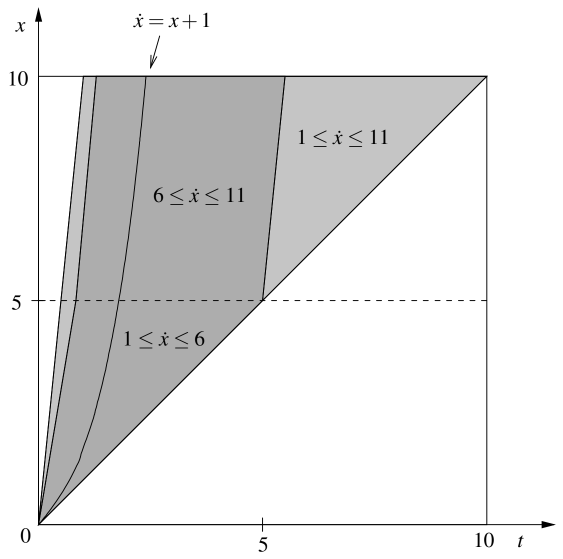


Fig. 12 Tighter approximations using evolution domain (invariant) splitting.

$[\![\phi]\!]_\varepsilon := \{v \in \mathbb{R}^X \mid \exists u \in [\![\phi]\!] : \max_{x \in X} |v(x) - u(x)| \le \varepsilon\}$ is the set of valuations at distance at most ε of a valuation satisfying ϕ.

It can be shown that for every hybrid automaton H and $\varepsilon > 0$, there exists a rectangular phase-portrait approximation H_ε of a splitting of H such that H_ε is an abstraction of H, and there exists ε-relaxation of H which is an abstraction of H_ε (see [93]). This ensures that if H robustly satisfies a safety property (i.e. both H and some ε-relaxation satisfy the safety property), then it is possible to establish the property using rectangular phase-portrait approximation and splitting.

In practice, it is often useful to split locations according to specific information we may have about the given hybrid automaton. For example, a flow constraint $\dot{x} = 3 - x$ suggests to split the evolution domain along lines parallel to $L \equiv 3 - x = 0$. More generally, a common heuristics is to use linear approximations of the flow constraints as support of cutting planes.

5.3 *Predicate Abstractions*

This part is a survey of [46, 10, 9]. We provide general ideas and guidelines about predicate abstraction schemes for hybrid systems. Chapter **??**16 provides a detailed presentation of abstraction techniques for program verification (note that imperative programs can be viewed as a subclass of hybrid systems).

Reachability analysis based on predicate abstraction consists of tracking the truth value of a fixed finite set of predicates instead of computing the value of the continuous variables. The continuous part of the state space is replaced by the Boolean truth values of the predicates.

Let H be a hybrid system, and let $\Pi = \{\pi_1, \dots, \pi_k\}$ be a finite set of linear predicates π_i of the form $y \bowtie 0$ where $y \in \mathsf{LTerm}(X)$ and $\bowtie \in \{<, \leq, =, >, \geq\}$. A truth value for Π is a vector $b \in \mathbb{B}^k$ where $\mathbb{B} = \{0,1\}$ that assigns a truth value b_i to each predicate $\pi_i \in \Pi$. Truth values induce a partition of the continuous state space into finitely many abstract states. To obtain an abstraction we require that whenever there exists a transition between two concrete states, then there is a transition between the corresponding abstract states. Hence, the transition relation satisfies Definition 11 by construction.

We define an abstraction mapping α_Π as follows. For all states (ℓ, v) of the hybrid automaton H, let $\alpha_\Pi(\ell, v) = (\ell, b)$ if $b = (b_1, \dots, b_k) \in \mathbb{B}^k$ is the vector of truth values of the predicates in Π under valuation v, i.e. such that $\pi_i(v) = b_i$ for all $1 \leq i \leq k$. We sometimes omit the location and write $\alpha_\Pi(v) = b$. We denote by γ_Π the *concretization function* such that $\gamma_\Pi(b) = \{v \in \mathbb{R}^X \mid \alpha_\Pi(v) = b\}$ for all $b \in \mathbb{B}^k$.

The *predicate abstraction* of H induced by Π is the finite-state transition system $H_\Pi = \langle S_\Pi, S_0, S_f, \Sigma, \rightarrow_\Pi \rangle$ where:

- $S = \{(\ell, b) \in \mathsf{Loc} \times \mathbb{B}^k \mid \exists v \in [\![\mathsf{Inv}(\ell)]\!] : \alpha_\Pi(\ell, v) = (\ell, b)\}$
- $S_0 = \{(\ell, b_0) \in S_\Pi \mid \exists v \in [\![\mathsf{Init}(\ell)]\!] : \alpha_\Pi(\ell, v) = (\ell, b_0)\}$;
- $S_f = \{(\ell, b_f) \in S_\Pi \mid \exists v \in [\![\mathsf{Final}(\ell)]\!] : \alpha_\Pi(\ell, v) = (\ell, b_f)\}$;
- $\Sigma = \mathsf{Lab} \cup \{\mathsf{time}\}$ where Lab is the alphabet of H;
- For each $\sigma \in \mathsf{Lab}$, the transition relation $\rightarrow_\Pi$ contains all tuples $((\ell, b), \sigma, (\ell', b'))$ such that $\exists e = (\ell, \sigma, \ell') \in \mathsf{Edg} \cdot \exists v \in \gamma_\Pi(b) \cdot \exists v' \in \gamma_\Pi(b') : (\ell, v) \xrightarrow{\sigma} (\ell', v')$; and the transition relation $\rightarrow_\Pi$ contains the tuples $((\ell, b), \mathsf{time}, (\ell', b'))$ such that $\ell' = \ell$ and $\exists r \geq 0 \cdot \exists v \in \gamma_\Pi(b) \cdot \exists v' \in \gamma_\Pi(b') : (\ell, v) \xrightarrow{r} (\ell, v')$.

While predicate abstractions are finite-state, their size can be of prohibitive computational cost. The number of states in H_Π is at most exponential in the number of predicates in Π. In practice though, many truth value vectors are not feasible (i.e., they have an empty concretization). For example, think of a set of $2k$ predicates over two variables x and y, where k predicates define a partition of the values for x (e.g., $x \leq 0$, $0 \leq x \leq 1$, and $1 \leq x$) and k predicates define a partition of the values for y. Then the number of feasible abstract state is at most k^2 rather than 2^{2k}. Note that this example would still give a number of abstract states exponential in the number of variables. The dimension of the space is a well-known source of computational complexity. The choice of predicates is thus very important to obtain precise approximations at the least cost. The initial set of predicates is usually chosen manually. Natural candidates are the predicates occurring in the hybrid automaton itself, like the evolution domains and jump conditions. Automatic construction and refinement of predicate abstractions is discussed in Sect. 5.4.

For reachability analysis, it is usually not necessary to construct the entire transition systems of the predicate abstractions, because many states may not be reachable. On-the-fly approaches are used to simultaneously construct and explore the abstraction. Starting from the initial states in the abstraction, the transitions to other abstract states are explored as and when they are computed. A classical strategy is to explore the discrete successors first (because they are less expensive to compute), and then the continuous successors for increasing amounts of time, as long as no new discrete transition is enabled.

Computing discrete successors. Discrete successors can be computed as follows. Given $b \in \mathbb{B}^k$, let $\Pi(b) = \bigwedge_{i|b_i=1} \pi_i \wedge \bigwedge_{i|b_i=0} \overline{\pi_i}$ be the constraint defining the abstract state b. A transition $e = (\ell, \sigma, \ell')$ is *enabled* in a state (ℓ, b) if $\mathsf{EN}(e) := [\![\exists X \cdot \Pi(b) \wedge \mathsf{Inv}(\ell) \wedge \mathsf{Jump}(e)]\!] \cap [\![\mathsf{Inv}(\ell')]\!] \neq \varnothing$. The successor states of (ℓ, b) by enabled transition e are the abstract states (ℓ', b') such that $R_k \neq \varnothing$ where $R_0 = \mathsf{EN}(e)$ and for all $1 \leq i \leq k$, if $b'_i = 1$ then $R_i = R_{i-1} \cap [\![\pi_i]\!]$, and if $b'_i = 0$ then $R_i = R_{i-1} \cap [\![\overline{\pi_i}]\!]$. A procedure for computing b' can easily be derived from this definition. Note that it may be that both $R_{i-1} \cap [\![\pi_i]\!] \neq \varnothing$ and $R_{i-1} \cap [\![\overline{\pi_i}]\!] \neq \varnothing$ hold, which would lead to set successively $b'_i = 1$ and $b'_i = 0$, and explore both cases. A simple optimization to this procedure is for each $1 \leq i \leq k$ to set $b'_i = 1$ beforehand if $\mathsf{EN}(e) \cap [\![\overline{\pi_i}]\!] = \varnothing$, and set $b'_i = 0$ if $\mathsf{EN}(e) \cap [\![\pi_i]\!] = \varnothing$. If one of the two cases holds, then the corresponding predicate can be skipped in the computation of R_i's.

Computing continuous successors. In general, the continuous successors are not computed exactly, even according to the abstract transition relation. This is due to the lack of exact algorithmic methods for solving differential equations. Note that this is a difficult problem even if the differential equations in flow constraints have closed-form solution, like in linear systems. Given $R \subseteq \mathbb{R}^X$ and location ℓ, we want to compute the set $\mathsf{Post}_C(\{\ell\} \times R)$ of continuous successor states as defined in Sect. 4.1, but over-approximations are sufficient for our purpose. This is consistent with the framework of abstraction (in the sense of Definition 11), but strictly speaking we are exploring in this way an over-approximation of the transition system H_Π defined above.

Optimizations. Various optimizations and heuristics have been defined and evaluated on many examples in the literature, see e.g. [46, 10, 9]. For example, when we discover that a new abstract state s is reachable as a continuous successor under some flow constraint, we do not need to explore the continuous successors of s under the same flow constraint (unless s is also reachable by some discrete transition). This may significantly prune the search through the abstract state space. The search can also be guided to discover unsafe reachable states as quickly as possible. Various exploration strategies have been defined, based on giving priority to the most promising states, according to some greedy measures. For example, such measures may estimate the distance of the current state to the unsafe state, such as the Euclidean distance between the valuation of the variables in the abstract state and in the unsafe states, or a discrete distance as the smallest number of discrete transitions necessary to reach the unsafe states, possibly taking into account the jump condition on the edges. Combination thereof are also possible [10]. Finally, as in program verification [90], it may be useful to maintain a set of predicates Π specific to each location, because certain predicates that are relevant in a location may not be useful in other locations.

5.4 Abstraction Refinement

The abstraction schemes presented in Sect. 5.1 and Sect. 5.3 may not be sufficient to establish safety of a system. In particular, we know that safety of the abstraction implies safety of the original system, but non-safety of the abstraction is inconclusive. The process of refinement consists in constructing abstractions that are tighter (or more detailed) than a given abstraction, in order to prove safety. If the refinement process repeatedly fails in proving safety, then one can reasonably conclude that even if the original system may be safe indeed, it should not be considered as acceptable because its correctness is not robust, a small deviation in the implementation of the system being able to cause violation of the safety requirement [155, 58]. Such considerations are used to stop the refinement process when a specified level of precision is reached [64, 46].

In general, if T^A is an abstraction of T, then a *refinement* T^B of T^A is an abstraction of T which is such that T^A is an abstraction of T^B.

In the case of splitting and phase-portrait approximations, refinements can be obtained by further splitting locations. For predicate abstractions, adding new predicates gives a refinement. We present one of the most popular framework to discover new predicates automatically, the counterexample-guided abstraction refinement (CEGAR) [47, 46]. A general framework of abstraction-refinement is presented in Chapter **??**.

Spurious counterexamples. When a predicate abstraction fails to establish safety, the analysis usually returns a witness path from an initial abstract state to a final abstract state. Such a path $\rho = q_0 \xrightarrow{\sigma_1} q_1 \ldots \xrightarrow{\sigma_n} q_n$ is a *spurious* counterexample if there exists no path $(\ell_0, v_0) \xrightarrow{\sigma_1} (\ell_1, v_1) \ldots \xrightarrow{\sigma_n} (\ell_n, v_n)$ in the original system such that $(\ell_i, v_i) \in \gamma_\Pi(q_i)$ for all $0 \le i \le n$. Clearly, if a counterexample is not spurious, then we can immediately conclude that the original system is not safe. We present a standard approach to check whether a counterexample is spurious [9].

To simplify the presentation, we assume in this section that every edge has a different label that identifies it uniquely. The successor operator is

$$\mathsf{Post}_\sigma(S) = \{(\ell', v') \mid \exists (\ell, v) \in S : (\ell, v) \xrightarrow{\sigma} (\ell', v')\},$$

where $\mathsf{Post}_{\mathsf{time}}(\cdot) = \mathsf{Post}_C(\cdot)$ is the one-step continuous successor operator. Similarly, the predecessor operator is

$$\mathsf{Pre}_\sigma(S) = \{(\ell, v) \mid \exists (\ell', v') \in S : (\ell, v) \xrightarrow{\sigma} (\ell', v')\}.$$

Let $R_0 = \gamma_\Pi(q_0) \cap \{(\ell, v) \mid v \in [\![\mathsf{Inv}(\ell)]\!]\}$, and $R_{i+1} = \mathsf{Post}_{\sigma_i}(R_i) \cap \gamma_\Pi(q_{i+1}) \cap \{(\ell, v) \mid v \in [\![\mathsf{Inv}(\ell)]\!]\}$ for all $i \geq 0$. The counterexample ρ is spurious iff $R_i = \varnothing$ for some $0 \leq i \leq n$. Note that over-approximations of $\mathsf{Post}_{\sigma_i}(\cdot)$ may suffice to show that a counterexample is spurious, but under-approximations are necessary to establish with certainty that a counterexample exists in the original system.

Refinement. Assume that the counterexample is spurious, and let $j \geq 0$ such that $R_j \neq \varnothing$ and $R_{j+1} = \varnothing$. Then it is easy to prove that $R_j \cap \mathsf{Pre}_{\sigma_{j+1}}(\gamma_\Pi(q_{j+1})) = \varnothing$. New predicates should be added to the set Π in order to rule out the counterexample. Since $R_j \cap \mathsf{Pre}_{\sigma_{j+1}}(\gamma_\Pi(q_{j+1})) = \varnothing$, we can search for a set of predicates which separates R_j and $\mathsf{Pre}_{\sigma_{j+1}}(\gamma_\Pi(q_{j+1}))$. A set $\hat{\Pi}$ of predicates *separates* two sets R and Q if for every truth value $b \in \mathbb{B}^{\hat{\Pi}}$, we have either $\gamma_{\hat{\Pi}}(b) \cap R = \varnothing$ or $\gamma_{\hat{\Pi}}(b) \cap Q = \varnothing$.

Note that to separate closed polyhedra, one simple linear constraint is always sufficient, but since reachable states (and in particular states reachable by continuous flow) are approximated by non-convex unions of polyhedra, several simple constraints may be necessary. Several methods have been developed to separate polyhedral sets, which are beyond the scope of this chapter. We refer to [9] for references and discussion.

In some case, spuriousness can be established by analyzing fragments of the counterexample [46], i.e. trying to show that a sub-sequence in the counter-example is not feasible in the original system. Spurious fragments of length 2 are called *locally infeasible* in [9] and defined as follows: $q_{i-1} \xrightarrow{\sigma_i} q_i \xrightarrow{\sigma_{i+1}} q_{i+1}$ is spurious if $\mathsf{Post}_{\sigma_i}(\gamma_\Pi(q_{i-1})) \cap \gamma_\Pi(q_i) \cap \mathsf{Pre}_{\sigma_{i+1}}(\gamma_\Pi(q_{i+1})) = \varnothing$. Refinement is computed as above using separating predicates.

Various forms of robustness have been considered for hybrid systems, which, basically work by not distinguishing between almost safe and almost unsafe hybrid systems so that incorrect answers from the analysis procedure are accepted for such borderline cases, but correct answers are required for clear-cut cases. Different notions of robustness have been considered successfully [64, 158, 157].

5.5 Approximate Bisimulations

For discrete systems, the relationships between systems can be described by the notions of language inclusion, simulation, and bisimulation. These concepts have been transposed to continuous and hybrid systems [85], and extended to take advantage of metrics over state spaces [78]. While traditional simulation and bisimulation relations require the output traces of related states to be identical, it suffices for metric relations that they are sufficiently close. This can be used to construct a discrete bisimilar quotient by discretizing the state space. The quotient is then amenable to verification and controller synthesis techniques for discrete systems [76].

We briefly sketch out the principle of approximate bisimulations in discrete time. Two states x_1, x_2 are in a *ε-bisimulation relation* if their output values are within distance ε and for every successor state x_1' of x_1, x_2 has a matching successor state

x_2' so that x_1' and x_2' are also in the relation. As a consequence, the output traces of two states in the relation will never be more than ε apart. Note that the definition coincides with classical bisimulation for $\varepsilon = 0$. It is generally hard to compute ε-bisimulation relations exactly, but (under some mild assumptions) one can define a Lyapunov-like *bisimulation function* that maps pairs of states to a nonnegative value, and whose sub-level sets are in an approximate bisimulation relation. The existence of bisimulation relations can be tied to certain types of stability (the tendency of the system to go to its equilibrium point). For example, a bisimulation function of a linear continuous system with dynamics $\dot{x} = Ax$ and output signal $y = Cx$ can be computed efficiently even for high dimensional systems by solving a set of linear matrix inequalities (LMI) of the form $M \geq C^{\mathsf{T}}C$ and $A^{\mathsf{T}}M + MA \leq 0$. The LMI always has a solution if the system is stable. Therefore, two stable linear systems are always ε-bisimilar, and an upper bound on ε can be computed. Note that approximate bisimulations can be used to relate continuous-time to discrete-time systems, continuous-valued to discrete-values systems, etc.

Verification by simulation. Bisimulation relations can also be used to verify bounded-horizon properties on bounded regions by computing a finite number of trajectories, a technique called *verification by simulation* [61, 102]. Here, the proximity measure of the bisimulation relation is combined with a robustness measure on temporal logic formulas. Given an initial state x_0 from which a trajectory satisfies a temporal formula to some measure, a bisimulation metric allows one to identify a neighborhood of initial states that all satisfy the same formula. This is possible since the bisimulation metric guarantees that all trajectories from the neighborhood (including all trajectories starting in x_0) remain sufficiently close together to satisfy the formula. Given a (dense) bounded region of initial states, it is, under suitable assumptions, possible to identify a finite subset of initial states whose trajectories are sufficient to show that the system satisfies a temporal logic formula [79]. A similar approach has been developed for embedded control software [112]. Together with the work on robustness mentioned in Sect. 5.4, these results demonstrate how stability and robustness can be used to simplify verification tasks.

6 Logic-based Verification

The working principle behind logic-based verification is to use logical formulas for characterizing some parts of the hybrid systems verification problem and solve this verification problem or subproblem entirely by checking corresponding logical formulas for validity. There are even verification techniques for hybrid systems that are entirely based on logic and proof [137, 143], which are beyond the scope of this chapter, however. In this section we survey the basic principles behind these approaches and show what kind of reasoning can be used to verify safety properties of hybrid systems or their parts by showing the validity of logical formulas.[7]

We survey a number of different approaches that represent the verification problem by various logical formulas or logical constraints:

1. Polynomial barrier certificates [154]
2. Equational certificates from templates [161, 159]
3. Differential invariants [136, 148, 144, 142]

These logic-based verification approaches further have in common that they argue by invariance and are based on variations of the work of Sophus Lie, of Darboux, or of Aleksandr Lyapunov. Differential invariants are based on Sophus Lie's 1867–1873 work on what are now called Lie derivatives and Lie groups. Equational certificates are based on Darboux's 1878 results [56] on a way to use Sophus Lie's approach. Barrier certificates are based on variations of Aleksandr Lyapunov's 1884–1892 work on a criterion for stability, which is used for safety instead [154]. The logic-based verification techniques for hybrid systems are complementary, so barrier certificates, equational templates, and differential invariants can be used together and also combined as abstractions with reachability analysis techniques.

Consider a location ℓ or a continuous evolution mode ℓ of a hybrid automaton with polynomial dynamics defined by polynomial differential equations. Let

$$\dot{x_1} = f_1(x), \ldots, \dot{x_n} = f_n(x)$$

be the polynomial differential equation system of the mode, which we abbreviate by the (vectorial) differential equation $\dot{x} = f(x)$. The mode ℓ has an invariant condition $\mathsf{Inv} \in \mathsf{PConstr}(X)$. What we want to understand about it in model checking of safety properties is whether the system will always stay in a safe region when it follows this continuous evolution mode starting from some initial region. We represent the desired initial region by a constraint $\mathsf{Init} \in \mathsf{PConstr}(X)$. Finally, we consider a constraint $\mathsf{Safe} \in \mathsf{PConstr}(X)$ defining the safe states for which we want to show that our system never leaves the set of states $[\![\mathsf{Safe}]\!]$ satisfying Safe.

Definition 12 (Continuous mode safety problem). Let $\dot{x} = f(x)$ be a (vectorial) differential equation, i.e., a polynomial differential equation system

$$\dot{x_1} = f_1(x), \ldots, \dot{x_n} = f_n(x)$$

for the system variables $X = \{x_1, \ldots, x_n\}$. A *continuous system* $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ consists of a constraint $\mathsf{Inv} \in \mathsf{PConstr}(X)$ for the invariant condition (or evolution domain restriction), and a constraint $\mathsf{Init} \in \mathsf{PConstr}(X)$ for the initial condition. We say that the continuous system $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ is *safe with respect to* constraint $\mathsf{Safe} \in \mathsf{PConstr}(X)$ iff for all $\delta \in \mathbb{R}^{\geq 0}$ and all continuously differentiable functions $\varphi : [0, \delta] \to \mathbb{R}^X$ with $\varphi(0) \in [\![\mathsf{Init}]\!]$ also satisfy $\varphi(\delta) \in [\![\mathsf{Safe}]\!]$ provided that $\dot{\varphi}(t) = f(\varphi(t))$ and $f(t) \in [\![\mathsf{Inv}]\!]$ for all $t \in [0, \delta]$. We also say that the continuous

[7] It should be noted that the other verification techniques surveyed in this chapter benefit from logic as well, for example in their representation of big sets of states using simple logical formulas.

system $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ *respects* Safe if $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ is safe with respect to property Safe.

Logic-based verification techniques provide easily checkable witnesses to verify that a continuous system $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ respects Safe. The immediate significance for model checking is that they induce abstractions that can be used to terminate reachability computation.

Lemma 6 (Logical abstraction). *Let* $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ *be a continuous system of a mode* $\ell \in \mathsf{Loc}$ *of a hybrid automaton. If* $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ *respects* Safe*, then*

$$\mathsf{post}_\ell(\llbracket \mathsf{Init} \rrbracket) \subseteq \llbracket \mathsf{Safe} \rrbracket$$

If the continuous system $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ of a mode $\ell \in \mathsf{Loc}$ of a hybrid automaton respects the desired safety property Safe, (continuous) reachability computation can be terminated for all states in any subset $P \subseteq \llbracket \mathsf{Init} \rrbracket$, because, by monotonicity, Lemma 6 then implies

$$\mathsf{post}_\ell(P) \subseteq \llbracket \mathsf{Safe} \rrbracket$$

In particular, notice that it is useful for fast reachability computation if we can identify big sets Init that make $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ respects Safe. These sets Init are often much bigger than the original initial sets from Definition 6.

The logic-based verification techniques mentioned above have in common that they provide easily checkable witnesses for the verification. They further enjoy the benefit that they can be used for highly nonlinear dynamics. The primary challenge in all cases is the need to first find the witnesses or their shape, which corresponds to the challenge of finding the right directions for support functions.

An interesting special case of the continuous safety problem from Definition 12 is the case where Init and Safe are the same formula F. If the continuous system $(F, \dot{x} = f(x), \mathsf{Inv})$ is safe with respect to F, then F is called a (safety) *invariant*. In that case, Lemma 6 implies

$$\mathsf{post}_\ell(\llbracket F \rrbracket) \subseteq \llbracket F \rrbracket$$

That is, the continuous system will never be able to leave F. Thus, without reachability computation, one can conclude that reachable sets that are within $\llbracket F \rrbracket$ will stay there forever.

Observe that, despite the similar name, there is a crucial difference between an invariant condition Inv of a continuous system (or a mode in a hybrid system) and a safety invariant F. The difference is that we need to verify whether F is a safety invariant, while we just assume that the system obeys the invariant condition Inv. That is why Inv is also called an evolution domain restriction, because it restricts the admissible evolution domain of the continuous system. So, Inv is part of the system model, yet F is part of a safety property that we verify for the system model.

One of many possible approaches to logic-based verification is the one that focuses on showing that a formula F is a global invariant of a hybrid automaton by showing that it is an invariant for each discrete transition and an invariant for each continuous transition of the automaton. The best case is if F is the safety property and turns out to be a global invariant of the system in this manner. This is generally somewhat overly simplistic, because the verification does not necessarily have to work with the same invariant F in all places so that multiple invariants need to be used instead. Nevertheless, having this simple example of a single global invariant in mind is a useful guiding principle for logic-based verification approaches.

Related arguments have also been used for invariant generation as abstraction techniques for abstract interpretation [169]. Based on decidability results for o-minimal hybrid automata [109], this includes invariant generation techniques for linear systems based on Gröbner basis computations [169] rather than based on quantifier elimination [109]. The case of (hyper-rectangle) box invariants has been discussed in more detail elsewhere [170].

6.1 Polynomial Barrier Certificates

The basic idea behind barrier certificates is to find a barrier separating good and bad states that we can easily show to be impenetrable by the continuous system dynamics. Barrier certificates were proposed for safety verification [154].

Theorem 4 (Weak barrier certificate [154]). *Let* $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ *be a continuous system with safety constraint* Safe*. If B is a* (weak) barrier certificate *for a continuous safety problem, i.e., a polynomial satisfying*

$$\begin{aligned} B(x) &\leq 0 \quad \textit{for all initial states } x \in [\![\mathsf{Init}]\!] \\ B(x) &> 0 \quad \textit{for all unsafe states } x \notin [\![\mathsf{Safe}]\!] \\ \frac{\partial B}{\partial x}(x) f(x) &\leq 0 \quad \textit{for all states } x \in [\![\mathsf{Inv}]\!] \end{aligned}$$

Then the continuous system $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ *respects* Safe.

Barrier certificates themselves can be defined for more general non-polynomial cases, but the conditions are generally not computable, when $f_i(x)$ and B are not polynomials or $\mathsf{Inv}, \mathsf{Init}$, and Safe are not polynomial constraints. The purpose of a barrier certificate is to separate safe from unsafe states in a way that initial states are safe, and the differential equations can easily be seen to never cross the barrier between safe and unsafe states.

The importance of barrier certificates comes from the fact that they reduce a reachability question (can we ever reach an unsafe state) by a simple check on the directional derivative $\frac{\partial B}{\partial x}(x) f(x)$ of the barrier certificate along the differential equation of the system.

It had originally been proposed [153] that barrier certificates only need to be checked on the boundary of the barrier and that it would be sufficient to check the third condition in Theorem 4 for all $x \in [\![\mathsf{Inv}]\!]$ with $B(x) = 0$:

$$\frac{\partial B}{\partial x}(x) f(x) \leq 0 \quad \text{for all states } x \in [\![\mathsf{Inv}]\!] \text{ with } B(x) = 0 \tag{16}$$

This condition is generally not strong enough and can lead to soundness issues as the following example shows.

Example 2. When using condition (16), it looks as if the differential equation $\dot{x} = 1$ always stayed in the region $\mathsf{Safe} \equiv x^2 \leq 0$ because condition (16) succeeds as follows:

$$\frac{\partial x^2}{\partial x} 1 = 2x \leq 0 \quad \text{for all states } x \text{ with } x^2 = 0$$

This, however, is counterfactual, because the system $\dot{x} = 1$ will, of course, leave region $x^2 \leq 0$. Thus, the condition (16) is unsound. The same issue occurs for a suggestion on how to extend this approach to Boolean combinations of inequalities [84]. A discussion under which assumptions the conditions can be restricted to such subsets without losing soundness can be found in the literature [154, 136, 144, 142].

Checking on the boundary is sound, however, if the condition (16) is modified to a strict inequality, instead of a weak inequality:

Theorem 5 (Strict barrier certificate [154]). *Let* $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ *be a continuous system with safety constraint* Safe. *If B is a* (strict) barrier certificate *for a continuous safety problem, i.e., a polynomial satisfying*

$$\begin{aligned} B(x) &\leq 0 \quad \textit{for all initial states } x \in [\![\mathsf{Init}]\!] \\ B(x) &> 0 \quad \textit{for all unsafe states } x \notin [\![\mathsf{Safe}]\!] \\ \frac{\partial B}{\partial x}(x) f(x) &< 0 \quad \textit{for all states } x \in [\![\mathsf{Inv}]\!] \textit{ with } B(x) = 0 \end{aligned}$$

Then the continuous system $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ *respects* Safe.

Search procedures for barrier certificates include approaches that choose a degree-bound for the barrier certificate $B(x)$ and then turn the conditions from Theorem 4 into a convex optimization problem, which can be solved efficiently [154]. A similar approach has been proposed for Theorem 5, but the optimization problem is then non-convex [154] so that optimizers can get stuck in local optima.

Barrier certificates can be extended to systems with disturbances and to switching diffusion systems [154]. We refer to the literature for a discussion of those generalizations and examples [154].

6.2 Equational Certificates

Equational certificates [161, 159] serve a purpose that has quite some similarity to barrier certificates. They were introduced [161] at the same time as barrier certificates [154], and later rephrased and generalized [159] similar to a matrix reformulation of that idea [123]. Equational certificates have been investigated earlier by Darboux in 1878 [56] for continuous systems not in the context of hybrid systems.

Like barrier certificates, the conditions of equational certificates make a reachability analysis superfluous, because they give a simple certificate showing a property of the system. One major difference of equational certificates compared to barrier certificates is that an equational certificate consists of a single polynomial *equation* $p(x) = 0$, while a barrier certificate consists of a single polynomial *inequality* $B(x) \leq 0$. The other major difference is the condition itself. It is an equational criterion, not using inequalities. Another minor difference is that an equational certificate $p(x) = 0$ shows invariance of the property $p(x) = 0$ instead of separating initial states from bad states. That is a minor difference, though, because Safe is an invariance property that can be read off from a barrier certificate that separates Init from $\neg\mathsf{Safe}$.

Theorem 6 (Equational certificates [161]). *Let $p(x)$ be a polynomial and let $(p(x) = 0, \dot{x} = f(x), \mathsf{Inv})$ be a continuous system. If there is a polynomial $g(x)$ such that*

$$\frac{\partial p(x)}{\partial x}(x) f(x) = g(x) p(x)$$

for all $x \in [\![\mathsf{Inv}]\!]$, then $(p(x) = 0, \dot{x} = f(x), \mathsf{Inv})$ respects $p(x) = 0$. In particular, $p(x) = 0$ is an invariant of $(p(x) = 0, \dot{x} = f(x), \mathsf{Inv})$.

The equational template approach for equational certificates [161] works as follows. The user chooses a template for the polynomial equation $p(x) = 0$ and the system then uses linear equation solving and/or Gröbner basis computations [38] to check whether the equational certificate condition from Theorem 6 hold. In general, the approach may use the decision procedures of quantifier elimination in real-closed fields [49] to handle the nonlinear real arithmetic.

Common special cases of equational certificates include those where only numbers or only 0 is chosen for the polynomial $g(x)$. It had been originally proposed informally [161] that it should also be sufficient in Theorem 6 to check

$$\frac{\partial p(x)}{\partial x}(x) f(x) = 0 \quad \text{for all } x \in [\![\mathsf{Inv}]\!] \text{ with } p(x) = 0 \tag{17}$$

This variation is generally not strong enough and can lead to soundness issues.

Example 3. When using condition (17), it may seem as if $x^2 = 0$ were an invariant of the differential equation $\dot{x} = 1$, because condition (17) succeeds as follows:

$$\frac{\partial x^2}{\partial x} 1 = 2x = 0 \quad \text{for all } x \text{ with } x^2 = 0$$

This, however, is counterfactual, because the system $\dot{x} = 1$ will, of course, falsify the safety condition $x^2 = 0$ right away. Thus, the condition (17) is an unsound variation of Theorem 6. We refer to the literature [136, 142] for a discussion of the conditions under which stronger assumptions can be assumed without losing soundness.

There are additional conditions on the system dynamics and p, however, under which the restriction (17) remains correct [142]. That line of research also identifies under which conditions equational templates and equational differential invariants are complete for verifying equational safety properties [142].

6.3 Differential Invariants and Logical Certificates

Differential invariants are a generalized form of logic-based witness techniques for hybrid systems and generalize equational certificates [161] and barrier certificates [153, 154]. Like equational certificates [161] a differential invariant can be an equation $p(x) = 0$. Like barrier certificates [153, 154], they can be inequalities like $p(x) \leq 0$. Differential invariants can be general logical formulas with propositional combinations of mixed equations, strict inequalities, and weak inequalities, and can be extended to contain quantifiers for distributed hybrid systems [138]. Differential invariants have been introduced in 2008 [136] and later refined to an automatic verification procedure that searches for differential invariants [148]. Further results about the theory of differential invariants can be found in the literature [144, 142].

Given a continuous system $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$, we want to check if it respects Safe. As a short notation, we say that the formula $\mathsf{Init} \rightarrow [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ *is valid*, if the continuous system $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ respects the safety condition Safe. That is, if that continuous system will alway stays in the region Safe when it follows differential equation $\dot{x} = f(x)$ restricted to the evolution domain region Inv and when started in any initial state satisfying Init. Even though more complex representations can be used, we assume $\mathsf{Init}, \mathsf{Safe}$, and Inv to be (semi-algebraic) polynomial constraints. A simple form corresponds to the case where Init and Safe are the same formula F. If $F \rightarrow [\dot{x} = f(x) \& \mathsf{Inv}]F$ is valid, then F is called a *continuous invariant* of the dynamics $\dot{x} = f(x) \& \mathsf{Inv}$. That is, if the continuous system starts in F, then it will always stay in F.

In fact, the notation $\mathsf{Init} \rightarrow [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ can be understood as a logical formula. The logical formula $[\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ uses the modal operator $[\dot{x} = f(x) \& \mathsf{Inv}]$ to say that formula Safe holds in all states that are reachable along the differential equation $\dot{x} = f(x)$ within revolution domain Inv. The implication $\mathsf{Init} \rightarrow$ in $\mathsf{Init} \rightarrow [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ restricts this to only the set of initial states that satisfy Init. The same principle extends to a logic for hybrid systems [135, 136, 137, 143, 141] and to a logic for distributed hybrid systems [140]; see [143] for an overview. Both of those logics are relatively complete (similarly to relative completeness of Hoare calculus). That is, they can prove every valid formula about hybrid systems or (distributed) hybrid systems from elementary properties of differential equations. Those results also give a precise construction lifting all verification techniques for continuous systems to hybrid systems [141].

Differential invariants can be equational formulas like equational certificates, they can include inequations like barrier certificates, but they also include mixed cases, Boolean combinations, and cases with more complicated logical formulas.

Definition 13 (Continuous invariant). Let $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ be a continuous system with safety constraint Safe. Constraint F is a *continuous invariant* of $\mathsf{Init} \rightarrow [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ iff the following formulas are valid (true in all states):

1. $\mathsf{Init} \wedge \mathsf{Inv} \rightarrow F$ (induction start), and
2. $F \rightarrow [\dot{x} = f(x) \& \mathsf{Inv}]F$ (induction step).

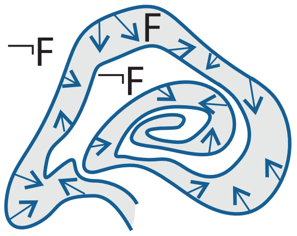


Fig. 13 Differential invariant F

A continuous invariant F is *sufficiently strong* for $\mathsf{Init} \rightarrow [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ if, in addition, $F \rightarrow \mathsf{Safe}$ is valid, because $\mathsf{Init} \rightarrow [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ is then valid.

It is easy to see that the existence of a sufficiently strong continuous invariant for $\mathsf{Init} \rightarrow [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ implies that the property $\mathsf{Init} \rightarrow [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ is valid.

Continuous invariants are useful notions, but they are not computational per se, because we still need to find a way to check the induction step. The induction start is reasonable, because it is just a constraint, which is a logical formula of first-order real arithmetic and thus decidable by quantifier elimination in real-closed fields [49, 166, 50]. But we need to find a checkable representation of the induction step. A checkable condition is made formally precise using the notion of differential invariants.

Definition 14 (Differential invariant). Let $(\mathsf{Init}, \dot{x} = f(x), \mathsf{Inv})$ be a continuous system with safety constraint Safe. A polynomial constraint F is a *differential invariant* of $\mathsf{Init} \rightarrow [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ iff the following formulas are valid:

1. $\mathsf{Init} \wedge \mathsf{Inv} \rightarrow F$ (induction start), and
2. $\mathsf{Inv} \rightarrow \nabla_{\dot{x}=f(x)} F$ (induction step),

where $\nabla_{\dot{x}=f(x)} F$ is the conjunction of all directional derivatives of atomic formulas in F in the direction of the vector field of $\dot{x} = f(x)$ (the partial derivative of b by x_i is $\frac{\partial b}{\partial x_i}$):

$$\nabla_{\dot{x}=f(x)} F \equiv \bigwedge_{(b \sim c) \in F} \left(\sum_{i=1}^{n} \frac{\partial b}{\partial x_i} f_i(x) \right) \sim \left(\sum_{i=1}^{n} \frac{\partial c}{\partial x_i} f_i(x) \right) \text{ where } \sim \in \{=, \geq, >, \leq, <\}.$$

A differential invariant F is *sufficiently strong* for $\mathsf{Init} \rightarrow [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ if, in addition, $F \rightarrow \mathsf{Safe}$ is valid (because $\mathsf{Init} \rightarrow [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ is then valid by Corollary 2 below).

The respective partial derivatives of terms are well-defined in the Euclidean space spanned by the variables and can be computed symbolically [136, 137]. Differential invariants capture the condition showing that the formula F is only becoming more true when following the dynamics, not less true, see Fig. 13.

The central property of differential invariants for verification purposes is that they replace infeasible or impossible reachability analysis with feasible symbolic computation.

Theorem 7 (Principle of differential induction [136]). *All differential invariants are continuous invariants.*

Corollary 1. *If F is a differential invariant for* $\mathsf{Init} \to [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$*, then* $\mathsf{Init} \to [\dot{x} = f(x) \& \mathsf{Inv}]F$ *is valid.*

Corollary 2. *If F is a differential invariant for* $\mathsf{Init} \to [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ *that is sufficiently strong, then F is a continuous invariant that is sufficiently strong for* $\mathsf{Init} \to [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$. *In particular,* $\mathsf{Init} \to [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$ *is valid.*

Example 4. Consider the dynamics $\dot{x} = x^4, \dot{y} = -2$. We are interested in seeing whether $2x \geq 5y$ is an invariant of this dynamics. With differential invariants it is easy to show that this is an invariant for the dynamics without using any state-based reachability verification. We just compute symbolically:

$$\nabla_{\dot{x}=x^4,\dot{y}=-2}(2x \geq 5y) \;\equiv\; \frac{\partial 2x}{\partial x}x^4 + \frac{\partial 2x}{\partial y}(-2) \geq \frac{\partial 5y}{\partial x}x^4 + \frac{\partial 5y}{\partial y}(-2) \;\equiv\; 2x^4 \geq -10.$$

Since the latter formula is easily found to be valid, $2x \geq 5y$ is proven to be a differential invariant and thus stays true whenever it holds for the initial state of the dynamics.

Consider the case where $\dot{x} = x^4, \dot{y} = -2$ is the dynamics of one location of a hybrid automaton. Then we know that $2x \geq 5y$ is true after staying in this location arbitrarily long, if only we know that $2x \geq 5y$ is also true initially when entering the location. This is a prototypical scenario where local verification results also need to be combined together in order to verify the whole hybrid automaton.

Example 5. Consider the dynamics

$$\dot{x_1} = 2x_1^4x_2 + 4x_1^2x_2^3 - 6x_1^2x_2, \dot{x_2} = -4x_1^3x_2^2 - 2x_1x_2^4 + 6x_1x_2^2$$

Using differential invariants it is easy to show that $x_1^4x_2^2 + x_1^2x_2^4 - 3x_1^2x_2^2 + 1 \leq c$ is an invariant of this dynamics, as illustrated in Fig. 14. The justification again follows by simple symbolic computation as in Example 4:

$$\begin{aligned}
&\nabla_{\dot{x_1}=2x_1^4x_2+4x_1^2x_2^3-6x_1^2x_2, \dot{x_2}=-4x_1^3x_2^2-2x_1x_2^4+6x_1x_2^2}(x_1^4x_2^2 + x_1^2x_2^4 - 3x_1^2x_2^2 + 1 \leq c)\\
&\equiv \frac{\partial(x_1^4x_2^2 + x_1^2x_2^4 - 3x_1^2x_2^2 + 1)}{\partial x_1}(2x_1^4x_2 + 4x_1^2x_2^3 - 6x_1^2x_2)\\
&\quad + \frac{\partial(x_1^4x_2^2 + x_1^2x_2^4 - 3x_1^2x_2^2 + 1)}{\partial x_2}(-4x_1^3x_2^2 - 2x_1x_2^4 + 6x_1x_2^2) \leq 0
\end{aligned}$$

which simplifies to true.

Differential invariants work somewhat like loop invariants but for differential equations instead of loops. When checking a loop invariant F, we can assume it holds before the loop in the induction step. It thus looks as if we should be able to assume F when proving the induction step Case 2 of Definition 14 and prove

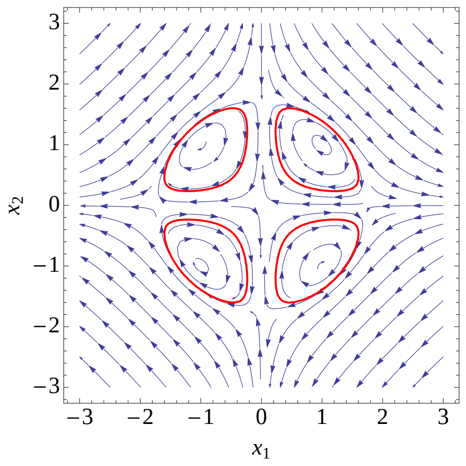


Fig. 14 Example of a differential invariant indicated by the thick boundary

$$\mathsf{Inv} \land F \to \nabla_{\dot{x}=f(x)} F \tag{18}$$

instead. Or, better, yet, only check the condition on the boundary of the domain like for barrier certificates. Neither of those would be sound, however, according to the following counterexamples from [136, 148]:

Example 6. When using condition (18), it looks as if $x^2 \leq 0$ were an invariant of the differential equation $\dot{x} = 1$, because condition (18) succeeds as follows:

$$(x^2 \leq 0 \to \nabla_{\dot{x}=1} x^2 \leq 0) \equiv (x^2 \leq 0 \to \frac{\partial x^2}{\partial x} 1 \leq 0) \equiv (x^2 \leq 0 \to 2x \leq 0)$$

This, however, is counterfactual, because the system $\dot{x} = 1$ will, of course, leave region $x^2 \leq 0$. Thus, the condition (18) is unsound. The same example shows that checking on the boundary of F is unsound in general. We refer to the original work [136] for a discussion of the conditions under which stronger assumptions can be assumed without losing soundness.

A further elaboration of those phenomena as well as an identification of the conditions under which such extra assumptions would be sound can be found in the literature [144, 142].

It turns out that some properties cannot be verified using differential invariants alone but that additional verification techniques are needed [144]. Differential saturation (repeated application of differential cuts [136, 144]) has been introduced together with differential invariants in 2008 [136] as a sound alternative that can be used to add conditions iteratively without compromising soundness.

Theorem 8 (Differential saturation [136, 144]). *Assume that F is a continuous invariant (e.g., a differential invariant) of* $\mathsf{Init} \to [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe}$*, then*

$$\mathsf{Init} \to [\dot{x} = f(x) \& \mathsf{Inv}]\mathsf{Safe} \quad \textit{iff} \quad \mathsf{Init} \to [\dot{x} = f(x) \& \mathsf{Inv} \land F]\mathsf{Safe}$$

An evolution domain constraint Inv (also confusingly referred to as invariant of a location) is an entirely different entity than an invariant property F of a system. An automaton model *assumes or prescribes* that the system dynamics can only be followed along traces that do not leave Inv, because the system will stop all executions that leave Inv. In contrast, a differential invariant *proves* that the system will never leave F whether it wants to or not. Nevertheless, Theorem 8 gives a sound way of translating a proved differential invariant into a prescriptive evolution domain constraint. Theorem 8 can be used to strengthen the evolution domain constraints to subregions, which then become available for subsequent verification in a sound way. The differential cut principle underlying Theorem 8 is particularly powerful when used repeatedly until saturation [136, 148]. That is, verification with differential invariants often proceeds in stages, where a number of formulas F are verified to be invariants and then used to constrain the evolution of the system using the right-hand side of Theorem 8. This process repeats until all unsafe states have been verified to be removable from the state space and so verification becomes trivial. Repeating this process in a fixpoint loop has been shown to work successfully in practice [148].

Differential invariants are computationally attractive concepts, because their induction start and induction step are just polynomial constraints, which are formulas of first-order real arithmetic, and are thus decidable by quantifier elimination in real-closed fields [49, 166, 50]. Also the check whether a differential invariant F is sufficiently strong to imply a polynomial safety constraint Safe is decidable. The steps needed to compute the induction step of a differential invariant are simple algebraic computations that can be automated easily.

Differential invariants are always sound. That is, every property that can be verified using a differential invariance is correct. The converse question is that of completeness, whether all relevant properties can be verified. It turns out that differential invariants alone are not complete.

Example 7. $x > 0 \to [\dot{x} = -x \& true]x > 0$ is valid, but $x > 0$ is not a differential invariant of $\dot{x} = -x$, not a barrier certificate, and does not qualify as an equational template either.

More generally, it can be shown that there are properties like Example 7 that are true but cannot be verified [144], except when using an additional verification technique known as differential auxiliaries (alias differential ghosts) that adds additional variables and additional dynamics for verification purposes [144]. Thus, differential auxiliaries are a fundamental extension that is required for verification.

Search procedures for differential invariants include degree-bounded enumeration and fixed point loops [148]. For completeness guarantees and numerous provability relationships on classes of differential invariants, see [141, 144]. The case of equational differential invariants is elaborated in [142], in which case differential invariants are a necessary and sufficient criterion for invariant functions according to a corresponding result by Lie.

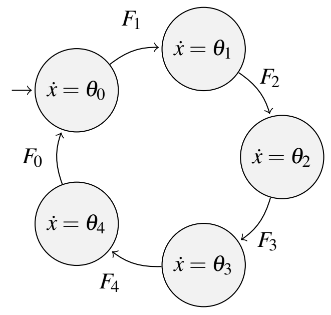


Fig. 15 Example of a verification loop for a hybrid automaton

Theorem 9 (Invariant function characterization). *A (polynomial) function p is an invariant function of $\dot{x} = f(x)$, i.e. the value of p along all solutions is constant, iff $p = 0$ is a differential invariant of $\dot{x} = f(x)$.*

A corresponding necessary and sufficient characterization of all algebraic invariant equations of algebraic differential equations is possible with a higher-order generalization of equational differential invariants called *differential radical invariants* [73].

For hybrid systems, differential invariants are used by allowing separate invariants for the respective locations of the hybrid automaton. Consider the hybrid automaton in Fig. 15 and, for the moment, suppose that there are no discrete jumps, i.e., the reset relations are the identity relation. Then, we need to show that starting in F_1 for dynamics $\dot{x} = \theta_1$ will always stay in the region F_2. In addition, we need to show that, when starting in F_2 the dynamics $\dot{x} = \theta_2$ will always stay in the region F_3, and so on. That is, in general we need to show that, when starting in F_i, the dynamics $\dot{x} = \theta_i$ will always stay in the region $F_{(i+1)\%5}$. In the presence of non-trivial discrete jump relations, we also need to show that those jump relations preserve the respective invariant. That means, we need to show that the jump relation (including its guard) will always transform every state within the invariant region F_i of its source into the invariant region $F_{(i+1)\%5}$ of its target. Finally, we only know that the reachable states of the hybrid automaton are contained in the respective invariant regions F_i if the automaton also starts in the required invariant region F_0 of the initial location. That is, we need to check that the initial state is contained in F_0.

To make this principle concrete, consider a flyable roundabout maneuver for air traffic control [149], which is a variation of roundabouts that have been proposed a decade before [175]. Flyable roundabouts follow a hybrid automaton similar to Fig. 15, but with locations that correspond to the various phases of the roundabout as depicted schematically in Fig. 16. The aircraft are initially in free flight (free), then, when a conflict arises, agree on a compatible roundabout collision avoidance maneuver (agree), approach the roundabout with an entry procedure (entry), follow the roundabout (circ), and then leave the roundabout (exit), until they are far enough away to enter free flight again. Such roundabout collision avoidance maneuvers for aircraft can be verified using differential invariants, see elsewhere [149] for details.

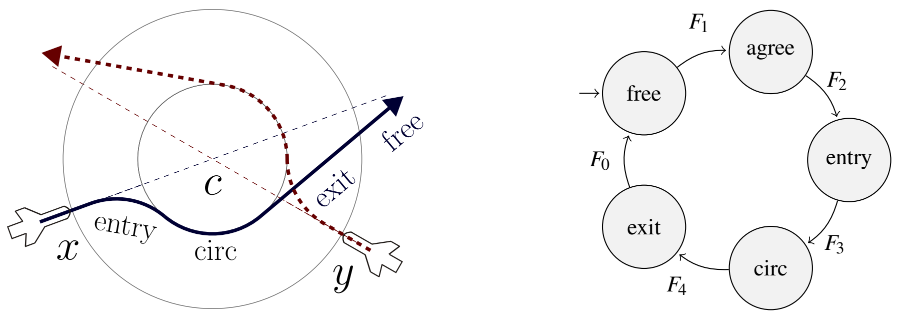


Fig. 16 Phases of flyable roundabout maneuver and protocol cycle

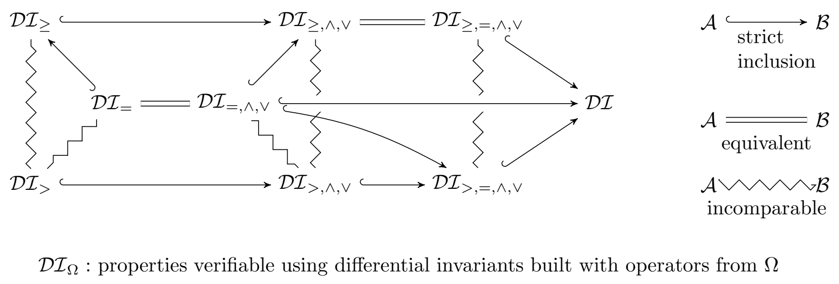

$\mathcal{DI}_{\geq}$
$\mathcal{DI}_{\geq,\wedge,\vee}$
$\mathcal{DI}_{\geq,=,\wedge,\vee}$
$\mathcal{DI}_{=}$
$\mathcal{DI}_{=,\wedge,\vee}$
$\mathcal{DI}$
$\mathcal{DI}_{>}$
$\mathcal{DI}_{>,\wedge,\vee}$
$\mathcal{DI}_{>,=,\wedge,\vee}$
$\mathcal{A} \hookrightarrow \mathcal{B}$ strict inclusion
$\mathcal{A} = \mathcal{B}$ equivalent
$\mathcal{A}$ $\mathcal{B}$ incomparable
$\mathcal{DI}_{\Omega}$: properties verifiable using differential invariants built with operators from Ω

Fig. 17 Differential invariance chart: identifies how classes of differential invariants relate to each other, where the operators in the differential invariants are restricted as indicated in subscript Ω

For an investigation of the theory of differential invariants, we refer to [136, 144, 142, 143, 73]. That line of research studies the theoretical and provability properties of differential invariants. It identifies a dozen relations either equating or separating the verification power of various classes of differential invariants (Fig. 17 indicates strict inclusion, equivalence, and incomparability of verification power, respectively). These relations further imply that the inclusion of Boolean operators that differential invariants support makes it possible to verify more systems compared to the single polynomial inequalities of barrier certificates or the single polynomial equations of equational templates [144]. The subclass of systems that have equation systems as invariants, however, already have a single equational invariant [144]. Differential cuts, differential saturation, and differential auxiliaries have been identified as fundamental extensions [136, 144]. The surprisingly close relationship of differential invariants to classical discrete invariants has been explored in the literature [141]. The relationship of differential invariants to Lie's seminal work, a differential operator view, and partial differential equations has been investigated along with a technique called inverse characteristic method for generating differential invariants [142]. The generalization of differential radical invariants can be generated efficiently using symbolic linear algebra [73].

For a generalization of differential invariants to systems with disturbances and differential-algebraic equations, we refer to the literature [136, 137]. Differential invariants can be generalized to the case of quantified first-order formulas and to distributed hybrid systems [138]. The approach extends to a relatively complete logic for hybrid systems [135, 136, 137, 141, 143] and to a relatively complete logic for distributed hybrid systems [140]. Generalizations to reachability and progress conditions can be found elsewhere [136]. Generalizations to stochastic hybrid systems with stochastic differential equations have been proposed [139].

7 Verification Tools

Despite the undecidability of the general case, the safety verification problem has been attacked algorithmically: many of the classical tools (*a.o.* HYTECH [89], CheckMate [45], D/DT [19]) and many of the more recent tools (PHAVER [67]) use a symbolic analysis of the hybrid automaton with a forward and/or backward approach: starting from the initial (resp. unsafe) states, iterate the Post operator (resp. Pre) until a fixed point is reached and then check emptiness of the intersection with the unsafe (resp. initial) states. By Theorem 2, those procedures are not guaranteed to terminate in general. As discussed in Sect. 4, a major issue is scalability, as the computational cost increases sharply with the number of continuous variables. Performance is achieved by overapproximating the Post operator, and overapproximation can also be used to force termination of the fixed point procedure. The challenge is to find methods that scale and are still accurate enough to show safety.

We discuss a selection of hybrid systems verification tools representing different classes of approaches that we survey here. A complete overview of all tools is beyond the scope of this article. We focus on a subset of the verification tools for which a dedicated tool paper and at least some documentation is available. A more complete collection of tools can be found on the Web[8].

HSolver: Interval Constraint Propagation

HSolver[9] [158] is an open-source software package for the formal verification of safety properties of continuous-time hybrid systems. It allows hybrid systems with non-linear ordinary differential equations and non-linear jumps assuming a global compact domain restriction on all variables. Even though HSolver is based on fast machine-precision floating point arithmetic, it uses sound rounding, and hence the correctness of its results cannot be hampered by round-off errors. HSolver not only verifies (unbounded horizon) reachability properties of hybrid systems, but—in addition—it also computes abstractions of the input system. So, even for input systems that are unsafe, or for which exhaustive formal verification is too difficult,

[8] `http://wiki.grasp.upenn.edu/`

[9] `http://hsolver.sourceforge.net/`

it will compute abstractions that can be used by other tools. For example, the abstractions could be used for guiding search for error trajectories of unsafe systems.

HSolver is not optimized for special classes of hybrid systems (e.g., systems such as linear hybrid automata that have very simple continuous dynamics). Moreover it does not yet provide mature support for finding counter-examples for unsafe input systems. The method used by HSolver is abstraction refinement based on interval constraint propagation [158], which incrementally refines an abstraction of the input system. Special care is taken to reflect as much information as possible into the abstraction without increasing its size.

HyTech: The HYbrid TECHnology Tool

HyTech[10] [89] was the first tool for reachability analysis of PCDA (Linear Hybrid Automata). The system is specified as a product of automata that synchronize on transitions that share the same label. The tool has a simple command language similar to a basic imperative language, allowing the user to program his own exploration algorithms. The basic data type represents union of polyhedra associated to each location of the product automaton. Operations such as Boolean operations, existential quantification, emptiness test, and reachability computation (using the Post operation) are provided. Error traces (counter examples) can be produced in combination with reachability analysis.

HyTech uses polyhedra with the double description method, which combines constraint and generator representations. The post operators are those described in Sect. 4.2 and implemented with exact arithmetic. HyTech can be used for parametric analysis by viewing parameters as variables with first derivative equal to zero. For instance, existential quantification on the reachable states can be used to extract a constraint on the parameters such that a given region is reachable. HyTech has been used to model check an audio control protocol [99] and a steam boiler [98]. A main limitation of HyTech lies in its use of standard integer data types, which quickly leads to integer overflow.

KeYmaera: Logic & Differential Invariants for Compositional Verification

KeYmaera[11] [135, 136, 137, 141, 150] is a hybrid verification tool for hybrid systems that combines deductive, real algebraic, and computer algebraic prover technologies. It is an automated and interactive theorem prover for a natural specification and verification logic for hybrid systems. With this, the verification principle behind KeYmaera is fundamentally different and complementary to tools like HyTech [89], PHAVer [67], and SpaceEx [68]. KeYmaera supports differential dynamic logic ($\mathsf{d}\mathcal{L}$) [135, 137, 141], which is a real-valued first-order dynamic logic

[10] `http://embedded.eecs.berkeley.edu/research/hytech/`

[11] `http://symbolaris.com/info/KeYmaera.html`

for hybrid programs [135, 137, 141], a program notation for hybrid systems. KeYmaera also supports hybrid systems with nonlinear discrete jumps, nonlinear differential equations, differential-algebraic equations, differential inequalities, and systems with nondeterministic discrete or continuous input.

For automation, KeYmaera implements a number of automatic proof strategies that decompose the hybrid system symbolically and prove the full system by proving properties of its parts [137]. This compositional verification principle helps scale up verification, because KeYmaera verifies large systems by verifying properties of subsystems (also see assume guarantee reasoning, 13). KeYmaera implements fixedpoint procedures [148] that compute differential invariants and invariants in fixedpoint loops, somewhat like classical model checkers compute reachable sets in fixedpoint loops. KeYmaera is typically more suitable for verifying parametric hybrid systems than systems with a single numerical state, where simulation is more appropriate. KeYmaera has been used successfully for verifying case studies in train control [151], car control [114, 115], air traffic management [149], mobile robotics [130] and surgical robotics [104]. The KeYmaera approach is described in a book about Logical Analysis of Hybrid Systems [137]. A comprehensive introduction is provided in a textbook on the Logical Foundations of Cyber-Physical Systems [146]. This textbook also explains how the successor tool KeYmaera X[12] [72] can achieve the same from a minimal soundness-critical core [145].

PHAVer: Polyhedral Hybrid Automaton Verifyer

PHAVer[13] [67] follows the same basic principles as HyTech. PHAVer is a formal verification tool for computing reachability and simulation relations of PCDA (Linear Hybrid Automata) from Sect. 4.2.

PHAVer uses standard operations on polyhedra for the reachability computation over an infinite time horizon (similar to those used in HyTech), and the algorithm for computing simulation relations is a straightforward extension of these. Using unbounded integer arithmetic, the computations are exact and formally sound. In addition to PCDA reachability, PHAVer can overapproximate piecewise affine dynamics on the fly, computing an overapproximation of the reachable states that is invariant (all trajectories that start within the set stay within the set). While PCDA are undecidable, PHAVer provides formally sound and precise overapproximation and widening operators that can force termination at the cost of reduced precision. These operators also simplify the computed continuous sets and dynamics of the system, and may result in a considerable speed-up without much loss in precision. The checking of abstraction and equivalence with simulation relations can be applied compositionally, and a sound non-circular assume-guarantee rule is implemented [66]. However, since the required exact computations on polyhedra do not scale well, this approach is limited to very small systems.

[12] `http://keymaeraX.org/`

[13] `http://www-verimag.imag.fr/~frehse/phaver_web/index.html`

With its exact computations and controllable overapproximations, PHAVer is suited for verifying formally stringent properties on small systems with simple dynamics, such as communication protocols with drifting clocks or buffer networks. PHAVer's disadvantage is that the employed polyhedra computations are generally exponential in the number of variables, so that scalability is limited. PHAVer has been used to verify oscillation properties of a voltage controlled oscillator circuit with 3 state variables [70], and various academic benchmarks with simple dynamics and up to 14 continuous variables. Since 2011, PHAVer is part of the SpaceEx tool platform [68].

SpaceEx: State Space Explorer

SpaceEx[14] [68] is a tool platform for verifying hybrid systems. It can handle hybrid automata whose continuous and jump dynamics are piecewise affine with nondeterministic inputs, i.e., PWA from Sect. 4.3. Nondeterministic inputs are particularly useful for modeling the approximation error when nonlinear systems are brought to piecewise affine form. SpaceEx comes with a compositional modeling language. It allows one to specify complex systems in a modular fashion as a network of interacting hybrid automata with templates and nesting. In the SpaceEx model editor, components are connected in block diagrams known from control theory, and the evolution of continuous variables is specified by hybrid automata with differential algebraic equations and inequalities.

Several different algorithms are implemented on the SpaceEx platform, including an exact algorithm for PCDA and a simulator that can handle nonlinear dynamics. The main verification algorithms, called LGG [68] and STC [69], combine explicit set representations (polyhedra), implicit set representations (support functions) and linear programming to achieve high scalability while maintaining high accuracy. The reachable states are overapproximated in the form of template polyhedra, which are polyhedra whose facets are oriented according to a user-provided set of template directions. The algorithms use adaptive time steps to ensure that the approximation error in each template direction remains below a given value. Empirical measurements indicate that the complexity of the image computations is linear in the number of variables, quadratic in the number of template directions, and linear in the number of time-discretization steps.

The accuracy of the overapproximation can be increased arbitrarily by choosing smaller time steps and adding more template directions. To attain a given approximation error (in the Hausdorff sense), the number of template directions is worst-case exponential. In case studies, the developers of SpaceEx observe that a linear number of user-specified directions, possibly augmented by a small set of critical directions, often suffices. The LGG and STC algorithms use floating-point computations that do not formally guarantee soundness. SpaceEx has been used to verify continuous and hybrid systems with more than 100 continuous variables.

[14] `http://spaceex.imag.fr/`

ToolboxLS: Level Set Methods

Level set methods are a class of algorithms designed for approximating the solution of the Hamilton-Jacobi partial differential equation (PDE) [127], which arises in many fields including optimal control, differential games, and dynamic implicit surfaces. In particular, dynamic implicit surfaces can be used to compute backward reachable sets and tubes for nonlinear, nondeterministic, continuous dynamic systems with control and/or disturbance inputs; in other words, inputs and parameters to the model can be treated in a worst case and/or best case fashion. The strengths and weaknesses of the Hamilton-Jacobi PDE formulation of reachability are very similar to those of viability theory: it can treat very general dynamics with adversarial inputs and can represent very general sets, but the known computational algorithms require resources that grow exponentially with the number of dimensions (number of variables); for example, in ToolboxLS[15] the level set algorithms run on a Cartesian grid of the state space. The ToolboxLS algorithms also do not guarantee the sign of computational errors, but they deliver higher accuracy for a given resolution than that available from typical sound alternatives.

Because ToolboxLS [128] is designed for dynamic implicit surfaces rather than specifically for reachability, it does not include a specialized verification interface; however, it has a 140 page user manual documenting the software and over twenty complete examples including three reachable set computations. It has been used primarily for reachability of systems with two to four continuous dimensions, including collision avoidance, quadrotor flips, aerial refueling, automated landing, and glidepath recapture.

Acknowledgments

The authors thank the anonymous reviewers of this chapter for helpful feedback, as well as David Henriques and João Martins for their proofreading help. We also thank Andrew Sogokon for suggesting the nice illustration shown in Fig. 14.

References

1. *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*. IEEE, 2012.
2. M. Althoff and B. Krogh. Reachability analysis of nonlinear differential-algebraic systems. *Automatic Control, IEEE Transactions on*, PP(99):1–1, 2013.
3. M. Althoff, B. H. Krogh, and O. Stursberg. Analyzing reachability of linear dynamic systems with parametric uncertainties. In A. Rauh and E. Auer, editors, *Modeling, Design, and Simulation of Systems with Uncertainties*. Springer, 2011.

[15] `http://www.cs.ubc.ca/~mitchell/ToolboxLS/`

4. Matthias Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 173–182. ACM, 2013.
5. Matthias Althoff and Bruce H Krogh. Avoiding geometric intersection operations in reachability analysis of hybrid systems. In *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, pages 45–54. ACM, 2012.
6. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
7. Rajeev Alur. Formal verification of hybrid systems. In Chakraborty et al. [41], pages 273–278.
8. Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Grossman et al. [82], pages 209–229.
9. Rajeev Alur, Thao Dang, and Franjo Ivancic. Counterexample-guided predicate abstraction of hybrid systems. *Theor. Comput. Sci.*, 354(2):250–271, 2006.
10. Rajeev Alur, Thao Dang, and Franjo Ivancic. Predicate abstraction for reachability analysis of hybrid systems. *ACM Trans. Embedded Comput. Syst.*, 5(1):152–199, 2006.
11. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
12. Rajeev Alur, Thomas Henzinger, Gerardo Lafferriere, and George J. Pappas. Discrete abstractions of hybrid systems. *Proc. IEEE*, 88(7):971–984, 2000.
13. Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. Software Eng.*, 22(3):181–201, 1996.
14. Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *ACM Symposium on Theory of Computing*, pages 592–601, 1993.
15. Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors, *HSCC*, volume 2034 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2001.
16. Rajeev Alur and George J. Pappas, editors. *Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proceedings*, volume 2993 of *LNCS*. Springer, 2004.
17. Eugene Asarin and Thao Dang. Abstraction by projection and application to multi-affine systems. In Alur and Pappas [16], pages 32–47.
18. Eugene Asarin, Thao Dang, and Antoine Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Inf.*, 43(7):451–476, 2007.
19. Eugène Asarin, Thao Dang, Oded Maler, and Olivier Bournez. Approximate reachability analysis of piecewise-linear dynamical systems. In *Proc. HSCC 00: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1790, pages 20–31. Springer-Verlag, 2000.
20. Eugene Asarin, Thao Dang, Oded Maler, and Romain Testylier. Using redundant constraints for refinement. In *Automated Technology for Verification and Analysis*, pages 37–51. Springer, 2010.
21. Eugene Asarin, Oded Maler, and Amir Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theor. Comput. Sci.*, 138(1):35–65, 1995.
22. R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.
23. Christel Baier, Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Marcus Größer. Almost-sure model checking of infinite paths in one-clock timed automata. In *Proc. of LICS*, pages 217–226. IEEE Computer Society, 2008.
24. Andrea Balluchi, Federico Di Natale, Alberto L. Sangiovanni-Vincentelli, and Jan H. van Schuppen. Synthesis for idle speed control of an automotive engine. In Alur and Pappas [16], pages 80–94.

25. Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc. of HSCC*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.
26. Alberto Bemporad, Antonio Bicchi, and Giorgio Buttazzo, editors. *Hybrid Systems: Computation and Control, 10th International Conference, HSCC 2007, Pisa, Italy, Proceedings*, volume 4416 of *LNCS*. Springer, 2007.
27. Alberto Bemporad and Manfred Morari. Verification of hybrid systems via mathematical programming. In Vaandrager and van Schuppen [178], pages 31–45.
28. Massimo Benerecetti, Marco Faella, and Stefano Minopoli. Automatic synthesis of switching controllers for linear hybrid systems: Safety control. *Theor. Comput. Sci.*, 493:116–138, July 2013.
29. Jan A. Bergstra and C. A. Middelburg. Process algebra for hybrid systems. *Theor. Comput. Sci.*, 335(2-3):215–280, 2005.
30. Martin Berz and Kyoko Makino. Verified integration of odes and flows using differential algebraic methods on high-order taylor models. *Reliable Computing*, 4(4):361–369, 1998.
31. A. Bicchi and L. Pallottino. On optimal cooperative conflict resolution for air traffic management systems. *IEEE Trans. Intelligent Transportation Systems*, 1(4):221–231, December 2000.
32. Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. Springer, Secaucus, NJ, USA, 1998.
33. Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, August 2004.
34. Michael S. Branicky. General hybrid dynamical systems: Modeling, analysis, and control. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems*, volume 1066, pages 186–200. Springer, 1995.
35. Michael S. Branicky, Vivek S. Borkar, and Sanjoy K. Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE T. Automat. Contr.*, 43(1):31–45, 1998.
36. T. Brihaye, L. Doyen, G. Geeraerts, J. Ouaknine, J.-F. Raskin, and J. Worrell. On reachability for hybrid automata over bounded time. In *Proceedings of ICALP 2011: International Colloquium on Automata, Languages and Programming (Part II)*, Lecture Notes in Computer Science 6756, pages 416–427. Springer-Verlag, 2011.
37. Lei Bu, You Li, Linzhang Wang, and Xuandong Li. Bach: Bounded reachability checker for linear hybrid automata. In *Formal Methods in Computer-Aided Design, 2008. FMCAD'08*, pages 1–4. IEEE, 2008.
38. Bruno Buchberger. *An Algorithm for Finding the Basis Elements of the Residue Class Ring of a Zero Dimensional Polynomial Ideal*. PhD thesis, University of Innsbruck, 1965.
39. Luca P. Carloni, Roberto Passerone, Alessandro Pinto, and Alberto L. Sangiovanni-Vincentelli. Languages and tools for hybrid systems design. *Foundations and Trends in Electronic Design Automation*, 1(1/2), 2006.
40. Franck Cassez and Kim Guldstrand Larsen. The impressive power of stopwatches. In *CONCUR*, pages 138–152, 2000.
41. Samarjit Chakraborty, Ahmed Jerraya, Sanjoy K. Baruah, and Sebastian Fischmeister, editors. *Proceedings of the 11th International Conference on Embedded Software, EMSOFT 2011, part of the Seventh Embedded Systems Week, ESWeek 2011, Taipei, Taiwan, October 9-14, 2011*. ACM, 2011.
42. Christopher Chase, Joseph Serrano, and Peter J Ramadge. Periodicity and chaos from switched flow systems: contrasting examples of discretely controlled continuous systems. *Automatic Control, IEEE Transactions on*, 38(1):70–83, 1993.
43. Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *RTSS*, pages 183–192. IEEE Computer Society, 2012.
44. N.V. Chernikova. Algorithm for discovering the set of all solutions of a linear programming problem. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968.

45. Alongkrit Chutinan and Bruce H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In Vaandrager and van Schuppen [178], pages 76–90.
46. Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Joël Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. J. Found. Comput. Sci.*, 14(4):583–604, 2003.
47. Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proc. of CAV 2000: Computer Aided Verification*, Lecture Notes in Computer Science 1855, pages 154–169. Springer-Verlag, 2000.
48. Pierre Collet and Jean Pierre Eckmann. *Iterated maps on the interval as dynamical systems*, volume 1. Springer, 1980.
49. George E. Collins. Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In H. Barkhage, editor, *Automata Theory and Formal Languages*, volume 33, pages 134–183. Springer, 1975.
50. George E. Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comput.*, 12(3):299–328, 1991.
51. Pieter Collins. Optimal semicomputable approximations to reachable and invariant sets. *Theory Comput. Syst.*, 41(1):33–48, 2007.
52. Pieter J. L. Cuijpers and Michel A. Reniers. Hybrid process algebra. *J. Log. Algebr. Program.*, 62(2):191–245, 2005.
53. Werner Damm, Hardi Hungar, and Ernst-Rüdiger Olderog. Verification of cooperating traffic agents. *International Journal of Control*, 79(5):395–421, May 2006.
54. Thao Dang and Romain Testylier. Reachability analysis for polynomial dynamical systems using the bernstein expansion. *Reliable Computing*, 17(2):128–152, 2012.
55. George B. Dantzig and B. Curtis Eaves. Fourier-motzkin elimination and its dual. *Journal of Combinatorial Theory*, 14:288–297, 1973.
56. Jean-Gaston Darboux. Mémoire sur les équations différentielles algébriques du premier ordre et du premier degré. *Bulletin des Sciences Mathématiques et Astronomiques*, 2(1):151–200, 1878.
57. M. De Wulf, L. Doyen, N. Markey, and J.-F. Raskin. Robust safety of timed automata. *Formal Methods in System Design*, 33(1-3):45–84, 2008.
58. Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP semantics: From timed models to timed implementations. *Formal Aspects of Computing*, 17(3):319–341, 2005.
59. L. Doyen. Robust parametric reachability for timed automata. *Information Processing Letters*, 102(5):208–213, 2007.
60. Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Automatic rectangular refinement of affine hybrid systems. In *Proc. of FORMATS 2005: Formal Modelling and Analysis of Timed Systems*, Lecture Notes in Computer Science 3829, pages 144–161. Springer-Verlag, 2005.
61. Georgios E Fainekos, Antoine Girard, and George J Pappas. Temporal logic verification using simulation. In *Formal Modeling and Analysis of Timed Systems*, pages 171–186. Springer, 2006.
62. Ansgar Fehnker and Bruce H. Krogh. Hybrid system verification is not a sinecure - the electronic throttle control case study. *Int. J. Found. Comput. Sci.*, 17(4):885–902, 2006.
63. Jeanne Ferrante and Charles Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM Journal on Computing*, 4(1):69–76, 1975.
64. Martin Fränzle. Analysis of hybrid systems: An ounce of realism can save an infinity of states. In *CSL*, Lecture Notes in Computer Science 1683, pages 126–140. Springer-Verlag, 1999.
65. Emilio Frazzoli and Radu Grosu, editors. *Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2011, Chicago, USA, April 12-14, 2011*. ACM, 2011.
66. Goran Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. PhD thesis, Radboud Universiteit Nijmegen, October 2005.

67. Goran Frehse. PHAVer: algorithmic verification of hybrid systems past HyTech. *STTT*, 10(3):263–279, 2008.
68. Goran Frehse, Colas Le Guernic, Alexandre Donzé, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV*, LNCS. Springer, 2011.
69. Goran Frehse, Rajat Kateja, and Colas Le Guernic. Flowpipe approximation and clustering in space-time. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 203–212. ACM, 2013.
70. Goran Frehse, Bruce H. Krogh, and Rob A. Rutenbar. Verifying analog oscillator circuits using forward/backward abstraction refinement. In Georges G. E. Gielen, editor, *DATE*, pages 257–262. European Design and Automation Association, Leuven, Belgium, 2006.
71. Robert M Freund and James B Orlin. On the complexity of four polyhedral set containment problems. *Mathematical programming*, 33(2):139–145, 1985.
72. Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In Amy Felty and Aart Middeldorp, editors, *CADE*, volume 9195 of *LNCS*, pages 527–538. Springer, 2015.
73. Khalil Ghorbal and André Platzer. Characterizing algebraic invariants by differential radical invariants. In Erika Ábrahám and Klaus Havelund, editors, *TACAS*, volume 8413 of *LNCS*, pages 279–294. Springer, 2014.
74. Pijush K Ghosh and K Vinod Kumar. Support function representation of convex bodies, its application in geometric computing, and some related representations. *Computer Vision and Image Understanding*, 72(3):379–403, 1998.
75. Antoine Girard. Reachability of uncertain linear systems using zonotopes. In Manfred Morari and Lothar Thiele, editors, *HSCC*, volume 3414 of *LNCS*, pages 291–305. Springer, 2005.
76. Antoine Girard. Controller synthesis for safety and reachability via approximate bisimulation. *Automatica*, 48(5):947–953, 2012.
77. Antoine Girard, Colas Le Guernic, and Oded Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In João P. Hespanha and Ashish Tiwari, editors, *HSCC*, volume 3927 of *LNCS*, pages 257–271. Springer, 2006.
78. Antoine Girard and George J Pappas. Approximation metrics for discrete and continuous systems. *Automatic Control, IEEE Transactions on*, 52(5):782–798, 2007.
79. Antoine Girard and Gang Zheng. Verification of safety and liveness properties of metric transition systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 11(S2):54, 2012.
80. Rafal Goebel, Ricardo G. Sanfelice, and Andrew R. Teel. Hybrid dynamical systems. *IEEE Control Systems Magazine*, 29(2):28–93, 2009.
81. Mark R Greenstreet. Verifying safety properties of differential equations. In *Computer Aided Verification*, pages 277–287. Springer, 1996.
82. Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors. *Hybrid Systems*, volume 736 of *LNCS*. Springer, 1993.
83. Colas Le Guernic and Antoine Girard. Reachability analysis of hybrid systems using support functions. In Ahmed Bouajjani and Oded Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 540–554. Springer, 2009.
84. Sumit Gulwani and Ashish Tiwari. Constraint-based approach for analysis of hybrid systems. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *LNCS*, pages 190–203. Springer, 2008.
85. Esfandiar Haghverdi, Paulo Tabuada, and George J Pappas. Bisimulation relations for dynamical, control, and hybrid systems. *Theoretical Computer Science*, 342(2):229–261, 2005.
86. Nicolas Halbwachs, Yann-Eric Proy, and Pascal Raymond. Verification of linear hybrid systems by means of convex approximations. In *International Static Analysis Symposium, SAS'94*, Namur (Belgium), September 1994.

87. T. A. Henzinger, Pei-Hsin Ho, and H. Wong-Toi. Hytech: the next generation. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS '95)*, page 56. IEEE Computer Society, 1995.
88. T.A. Henzinger. Hybrid automata with finite bisimulations. In *ICALP: Automata, Languages, and Programming*, Lecture Notes in Computer Science 944, pages 324–335. Springer, 1995.
89. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, pages 110–122, 1997.
90. T.A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *Proceedings of the 29th Annual Symposium on Principles of Programming Languages*, pages 58–70. ACM Press, 2002.
91. Thomas A. Henzinger. The theory of hybrid automata. In M.K. Inan and R.P. Kurshan, editors, *Verification of Digital and Hybrid Systems*, NATO ASI Series F: Computer and Systems Sciences 170, pages 265–292. Springer-Verlag, 2000.
92. Thomas A Henzinger and Pei-Hsin Ho. A note on abstract interpretation strategies for hybrid automata. In *Hybrid Systems II*, pages 252–264. Springer, 1995.
93. Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.
94. Thomas A. Henzinger and Peter W. Kopke. State equivalences for rectangular hybrid automata. In *CONCUR: Concurrency Theory*, Lecture Notes in Computer Science 1119, pages 530–545. Springer, 1996.
95. Thomas A. Henzinger and Peter W. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221:369–392, 1999.
96. Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.
97. Thomas A. Henzinger and Jean-François Raskin. Robust undecidability of timed and hybrid systems. In *Proc. of HSCC 00: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1790, pages 145–159. Springer-Verlag, 2000.
98. Thomas A. Henzinger and Howard Wong-Toi. Using hytech to synthesize control parameters for a steam boiler. In Jean-Raymond Abrial, Egon Börger, and Hans Langmaack, editors, *Formal Methods for Industrial Applications*, volume 1165 of *Lecture Notes in Computer Science*, pages 265–282. Springer, 1995.
99. Pei-Hsin Ho and Howard Wong-Toi. Automated analysis of an audio control protocol. In Pierre Wolper, editor, *CAV*, volume 939 of *Lecture Notes in Computer Science*, pages 381–394. Springer, 1995.
100. Taylor T. Johnson and Sayan Mitra. Parametrized verification of distributed cyber-physical systems: An aircraft landing protocol case study. In *ICCPS*, pages 161–170. IEEE, 2012.
101. Hossein Jula, Elias B. Kosmatopoulos, and Petros A. Ioannou. Collision avoidance analysis for lane changing and merging. PATH Research Report UCB-ITS-PRR-99-13, Institute of Transportation Studies, University of California, Berkeley, 1999.
102. A Agung Julius, Georgios E Fainekos, Madhukar Anand, Insup Lee, and George J Pappas. Robust test generation and coverage for hybrid systems. In *Hybrid Systems: Computation and Control*, pages 329–342. Springer, 2007.
103. Kyoung-Dae Kim and P. R. Kumar. Cyber-physical systems: A perspective at the centennial. *Proceedings of the IEEE*, 100(Centennial-Issue):1287–1308, 2012.
104. Yanni Kouskoulas, David W. Renshaw, André Platzer, and Peter Kazanzides. Certifying the safe design of a virtual fixture control algorithm for a surgical robot. In Calin Belta and Franjo Ivancic, editors, *HSCC*, pages 263–272. ACM, 2013.
105. Wolfgang Kühn. Rigorously computed orbits of dynamical systems without the wrapping effect. *Computing*, 61(1):47–67, 1998.
106. Alex A Kurzhanskiy and Pravin Varaiya. Ellipsoidal toolbox (et). In *Decision and Control, 2006 45th IEEE Conference on*, pages 1498–1503. IEEE, 2006.
107. Alex A Kurzhanskiy and Pravin Varaiya. Ellipsoidal techniques for reachability analysis of discrete-time linear systems. *Automatic Control, IEEE Transactions on*, 52(1):26–38, 2007.
108. Gerardo Lafferriere, George J. Pappas, and Shankar Sastry. O-minimal hybrid systems. *Mathematics of Control, Signals, and Systems*, 13(1):1–21, March 2000.

109. Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. Symbolic reachability computation for families of linear vector fields. *J. Symb. Comput.*, 32(3):231–253, 2001.
110. Colas Le Guernic. *Reachability analysis of hybrid systems with linear continuous dynamics*. PhD thesis, Université Grenoble 1 - Joseph Fourier, 2009.
111. Edward Ashford Lee and Sanjit Arunjumar Seshia. *Introduction to Embedded Systems — A Cyber-Physical Systems Approach*. Lulu.com, 2013.
112. Flavio Lerda, James Kapinski, Edmund M Clarke, and Bruce H Krogh. Verification of supervisory control software using state proximity and merging. In *Hybrid Systems: Computation and Control*, pages 344–357. Springer, 2008.
113. Carolos Livadas, John Lygeros, and Nancy A. Lynch. High-level modeling and analysis of TCAS. *Proc. IEEE - Special Issue on Hybrid Systems: Theory & Applications*, 88(7):926–947, 2000.
114. Sarah M. Loos, André Platzer, and Ligia Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In Michael Butler and Wolfram Schulte, editors, *FM*, volume 6664 of *LNCS*, pages 42–56. Springer, 2011.
115. Sarah M. Loos, David Witmer, Peter Steenkiste, and André Platzer. Efficiency analysis of formally verified adaptive cruise controllers. In Andreas Hegyi and Bart De Schutter, editors, *ITSC*. Springer, 2013.
116. Alexander V. Lotov, Vladimir A. Bushenkov, and Georgy K. Kamenev. *Interactive Decision Maps*, volume 89 of *Applied Optimization*. Kluwer Academic Publishers, Boston, 2004.
117. Jan Lunze and Françoise Lamnabhi-Lagarrigue. *Handbook of hybrid systems control: theory, tools, applications*. Cambridge University Press, 2009.
118. Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Hybrid I/O automata. *Inf. Comput.*, 185(1):105–157, 2003.
119. Oded Maler. Algorithmic verification of continuous and hybrid systems. In *Int. Workshop on Verification of Infinite-State System (Infinity)*, 2013.
120. Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX Workshop*, volume 600, pages 447–484. Springer, 1991.
121. Oded Maler and Amir Pnueli, editors. *Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3-5, 2003, Proceedings*, volume 2623 of *LNCS*. Springer, 2003.
122. Peter Marwedel. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*. Springer, 2010.
123. Nadir Matringe, Arnaldo Vieira Moura, and Rachid Rebiha. Generating invariants for non-linear hybrid systems by linear algebraic methods. In Radhia Cousot and Matthieu Martel, editors, *SAS*, volume 6337 of *Lecture Notes in Computer Science*, pages 373–389. Springer, 2010.
124. Joseph S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In *Proc. of HSCC 00: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1790, pages 296–309. Springer-Verlag, 2000.
125. Robin Milner. An algebraic definition of simulation between programs. In David C. Cooper, editor, *Proc. of the 2nd Int. Joint Conference on Artificial Intelligence. London, UK, September 1971*, pages 481–489. William Kaufmann, British Computer Society, 1971.
126. Robin Milner. *A Calculus of Communicating Systems*. Lecture Notes in Computer Science 92. Springer-Verlag, 1980.
127. Ian M. Mitchell, Alexandre M. Bayen, and Claire J. Tomlin. A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. *IEEE T. Autom. Contr.*, 50(7):947–957, 2005.
128. Ian M. Mitchell and Jeremy A. Templeton. A toolbox of Hamilton-Jacobi solvers for analysis of nondeterministic continuous and hybrid systems. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, number 3414 in LNCS, pages 480–494. Springer Verlag, 2005.

129. Sayan Mitra, Yong Wang, Nancy A. Lynch, and Eric Feron. Safety verification of model helicopter controller using hybrid input/output automata. In Maler and Pnueli [121], pages 343–358.
130. Stefan Mitsch, Khalil Ghorbal, and André Platzer. On provably safe obstacle avoidance for autonomous robotic ground vehicles. In *Robotics: Science and Systems*, 2013.
131. A. Nerode and A. Yakhnis. Modelling hybrid systems as games. In *Decision and Control, 1992., Proceedings of the 31st IEEE Conference on*, pages 2947–2952 vol.3, 1992.
132. Anil Nerode and Wolf Kohn. Models for hybrid systems: Automata, topologies, controllability, observability. In Grossman et al. [82], pages 317–356.
133. Arnold Neumaier. The wrapping effect, ellipsoid arithmetic, stability and confidence regions. In *Validation Numerics*, pages 175–190. Springer, 1993.
134. Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. An approach to the description and analysis of hybrid systems. In Grossman et al. [82], pages 149–178.
135. André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008.
136. André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, 20(1):309–352, 2010. Advance Access published on November 18, 2008.
137. André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010.
138. André Platzer. Quantified differential invariants. In Frazzoli and Grosu [65], pages 63–72.
139. André Platzer. Stochastic differential dynamic logic for stochastic hybrid programs. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *LNCS*, pages 431–445. Springer, 2011.
140. André Platzer. A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. *Logical Methods in Computer Science*, 8(4):1–44, 2012. Special issue for selected papers from CSL'10.
141. André Platzer. The complete proof theory of hybrid systems. In *LICS* [1], pages 541–550.
142. André Platzer. A differential operator approach to equational differential invariants. In Lennart Beringer and Amy Felty, editors, *ITP*, volume 7406 of *LNCS*, pages 28–48. Springer, 2012.
143. André Platzer. Logics of dynamical systems. In *LICS* [1], pages 13–24.
144. André Platzer. The structure of differential invariants and differential cut elimination. *Logical Methods in Computer Science*, 8(4):1–38, 2012.
145. André Platzer. A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reas.*, 59(2):219–265, 2017.
146. André Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, Switzerland, 2017.
147. André Platzer and Edmund M. Clarke. The image computation problem in hybrid systems model checking. In Bemporad et al. [26], pages 473–486.
148. André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. *Form. Methods Syst. Des.*, 35(1):98–120, 2009.
149. André Platzer and Edmund M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In Ana Cavalcanti and Dennis Dams, editors, *FM*, volume 5850 of *LNCS*, pages 547–562. Springer, 2009.
150. André Platzer and Jan-David Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 171–178. Springer, 2008.
151. André Platzer and Jan-David Quesel. European Train Control System: A case study in formal verification. In Karin Breitman and Ana Cavalcanti, editors, *ICFEM*, volume 5885 of *LNCS*, pages 246–265. Springer, 2009.
152. Pavithra Prabhakar and Mahesh Viswanathan. A dynamic algorithm for approximate flow computations. In Frazzoli and Grosu [65], pages 133–142.
153. Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In Alur and Pappas [16], pages 477–492.

154. Stephen Prajna, Ali Jadbabaie, and George J. Pappas. A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE T. Automat. Contr.*, 52(8):1415–1429, 2007.
155. Anuj Puri. Dynamical properties of timed automata. In *FTRTFT '98*, Lecture Notes in Computer Science 1486, pages 210–227. Springer-Verlag, 1998.
156. Anuj Puri and Pravin Varaiya. Decidability of hybrid systems with rectangular differential inclusion. In *Proc. of CAV*, volume 818 of *Lecture Notes in Computer Science*, pages 95–104. Springer, 1994.
157. Stefan Ratschan. Safety verification of non-linear hybrid systems is quasi-semidecidable. In *TAMC 2010: 7th Annual Conference on Theory and Applications of Models of Computation*, volume 6108 of *LNCS*, pages 397–408. Springer, 2010.
158. Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *Trans. on Embedded Computing Sys.*, 6(1):8, 2007.
159. Sriram Sankaranarayanan. Automatic invariant generation for hybrid systems using ideal fixed points. In Karl Henrik Johansson and Wang Yi, editors, *HSCC*, pages 221–230. ACM, 2010.
160. Sriram Sankaranarayanan, Thao Dang, and Franjo Ivančić. Symbolic model checking of hybrid systems using template polyhedra. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 188–202. Springer, 2008.
161. Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Constructing invariants for hybrid systems. *Formal Methods in System Design*, 32(1):25–55, 2008.
162. Marc Segelken. Abstraction and counterexample-guided construction of *omega* -automata for model checking of step-discrete linear hybrid models. In Werner Damm and Holger Hermanns, editors, *CAV*, volume 4590 of *LNCS*, pages 433–448. Springer, 2007.
163. Oleg Sokolsky, Insup Lee, and Mats Per Erik Heimdahl. Challenges in the regulatory approval of medical cyber-physical systems. In Chakraborty et al. [41], pages 227–232.
164. Olaf Stursberg, Ansgar Fehnker, Zhi Han, and Bruce H. Krogh. Verification of a cruise control system using counterexample-guided search. *Control Engineering Practice*, 2004.
165. Paulo Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.
166. Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, 2nd edition, 1951.
167. Lucio Tavernini. Differential automata and their discrete simulators. *Non-Linear Anal.*, 11(6):665–683, 1987.
168. Ashish Tiwari. Approximate reachability for linear systems. In Maler and Pnueli [121], pages 514–525.
169. Ashish Tiwari. Abstractions for hybrid systems. *Form. Methods Syst. Des.*, 32(1):57–83, 2008.
170. Ashish Tiwari. Generating box invariants. In Magnus Egerstedt and Bud Mishra, editors, *HSCC*, volume 4981 of *LNCS*, pages 658–661. Springer, 2008.
171. Ashish Tiwari. Logic in software, dynamical and biological systems. In *LICS*, pages 9–10. IEEE Computer Society, 2011.
172. Ashish Tiwari, Natarajan Shankar, and John M. Rushby. Invisible formal methods for embedded control systems. *Proc. IEEE*, 91(1):29–39, 2003.
173. Hans Raj Tiwary. On the hardness of computing intersection, union and minkowski sum of polytopes. *Discrete & Computational Geometry*, 40(3):469–479, 2008.
174. Claire Tomlin, George Pappas, Jana Košecká, John Lygeros, and Shankar Sastry. Advanced air traffic automation: A case study in distributed decentralized control. In Bruno Siciliano and Kimon Valavanis, editors, *Control Problems in Robotics and Automation*, volume 230 of *Lecture Notes in Control and Information Sciences*, pages 261–295. Springer, 1998.
175. Claire Tomlin, George J. Pappas, and Shankar Sastry. Conflict resolution for air traffic management: a study in multi-agent hybrid systems. *IEEE T. Automat. Contr.*, 43(4):509–521, 1998.

176. Shinya Umeno and Nancy A. Lynch. Proving safety properties of an aircraft landing protocol using I/O automata and the PVS theorem prover: A case study. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM*, volume 4085, pages 64–80. Springer, 2006.
177. Shinya Umeno and Nancy A. Lynch. Safety verification of an aircraft landing protocol: A refinement approach. In Bemporad et al. [26], pages 557–572.
178. Frits W. Vaandrager and Jan H. van Schuppen, editors. *Hybrid Systems: Computation and Control, Second International Workshop, HSCC'99, Berg en Dal, The Netherlands, March 29-31, 1999, Proceedings*, volume 1569 of *LNCS*. Springer, 1999.
179. D. A. van Beek, Ka L. Man, Michel A. Reniers, J. E. Rooda, and Ramon R. H. Schiffelers. Syntax and consistent equation semantics of hybrid Chi. *J. Log. Algebr. Program.*, 68(1-2):129–210, 2006.
180. D. A. van Beek, Michel A. Reniers, Ramon R. H. Schiffelers, and J. E. Rooda. Concrete syntax and semantics of the compositional interchange format for hybrid systems. In *17th IFAC World Congress*, 2008.
181. Howard Wong-Toi. Analysis of slope-parametric rectangular automata. In *Hybrid Systems*, LNCS 1567, pages 390–413. Springer, 1997.
182. Tichakorn Wongpiromsarn, Sayan Mitra, Richard M. Murray, and Andrew G. Lamperski. Periodically controlled hybrid systems. In Rupak Majumdar and Paulo Tabuada, editors, *HSCC*, volume 5469 of *LNCS*, pages 396–410. Springer, 2009.