HAL
open science

Scalable Verification of Hybrid Systems

Goran Frehse

To cite this version:

Goran Frehse. Scalable Verification of Hybrid Systems. Systems and Control [cs.SY]. Univ. Grenoble Alpes, 2016. tel-01714428

HAL Id: tel-01714428

https://hal.science/tel-01714428v1

Submitted on 21 Feb 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Communauté UNIVERSITÉ Grenoble Alpes

HABILITATION À DIRIGER DES RECHERCHES

Spécialité : **Informatique**

Présentée par

Goran Frehse

préparée au sein du laboratoire **Verimag** et de l'**Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Scalable Verification of Hybrid Systems

Habilitation à diriger des recherches soutenue publiquement le **26.05.2016**, devant le jury composé de :

M. Alain Girault
Directeur de Recherches, INRIA, Président
M. Eugene Asarin
Professeur, IRIF, Université Paris Diderot et CNRS, Rapporteur
M. Manfred Morari
Professeur, ETH Zurich, Suisse, Rapporteur
M. Pravin Varaiya
Professeur, University of California Berkeley, USA, Rapporteur
M. Rajeev Alur
Professeur, University of Pennsylvania, USA, Examinateur
M. Eric Goubault
Professeur, École polytechnique, Examinateur
M. Oded Maler
Directeur de Recherches, CNRS, Examinateur

Contents

1 Introduction **1**
1.1 Contributions . . . 3
1.2 Research Supervision and Projects . . . 4

2 Hybrid Automata and Reachability **7**
2.1 Hybrid Automata . . . 7
2.2 Set-based Reachability . . . 13
2.3 Reachability for Piecewise Affine Dynamics . . . 14

3 Template Reachability with Support Functions **21**
3.1 Support Functions and Template Polyhedra . . . 21
3.2 Flowpipe Approximation . . . 23
3.3 Computing Transition Successors . . . 28
3.4 Clustering . . . 30
3.5 Experimental Results . . . 31

4 One-Step Template Refinement **37**
4.1 Intersection of Support Functions with Polyhedra . . . 39
4.2 Flowpipe-Guard Intersection . . . 44
4.3 Branch-and-Bound Clustering . . . 46
4.4 Experimental Results . . . 47

5 Semi-Template Reachability in Space-Time **51**
5.1 Flowpipe Approximation in Space-Time . . . 55
5.2 Clustering in Space-Time . . . 64
5.3 Experimental Results . . . 71

6 Property-Based Template Refinement **75**
6.1 Approximate Support Functions . . . 76
6.2 Separating Convex Sets . . . 78
6.3 Timed Flowpipe Separation . . . 84
6.4 Experimental Results . . . 89

7 The SpaceEx Verification Platform **93**
7.1 SpaceEx Architecture . . . 96
7.2 Scenario Implementations . . . 99
7.3 Modeling in SpaceEx . . . 100

8 Conclusions and Outlook **105**

Chapter 1

Introduction

Hybrid systems describe the change of a set of continuously evolving, real-valued variables combined with discrete states. In continuous time, the change is governed by a set of ordinary differential equations (ODEs) or, more generally, inclusions. Jumps of the discrete state can modify the ODEs or the values of the variables, and such changes can be state-dependent and non-deterministic. Cyber-physical systems can be considered as hybrid systems if one can abstract from networking and distributed aspects. Hybrid systems occur in a wide range of domains such as automotive control, robotics, electronic circuits, systems biology, and health care. Hybrid automata are a modeling paradigm for hybrid systems. The discrete states and the possible transitions from one state to another are described with a finite state-transition system. Each discrete state is associated with a set of ODEs that describes how variables evolve with time. A change in discrete state can update the continuous variables. Hybrid automata are non-deterministic, which means that different futures may be available from any given state. Rates of change or variable updates can be described by providing bounds instead of fixed numbers. Incomplete knowledge about initial conditions, perturbations, parameters, etc. can easily be captured this way. Complex models are readily constructed by composing automata that interact by sharing variables and synchronizing events. Hybrid automata are well-suited for formal analysis, since sets of behaviors are readily described by mathematical equations. In this sense, they are an analysis-friendly rather than a user-friendly modeling formalism. There are no mechanisms to avoid modeling mistakes, such as unintended deadlocks.

The traditional technique for analyzing the behavior of hybrid systems, e.g., in model based design, is numerical simulation. Numerical simulation approximates the evolution of the variables with a sequence of points in discretized time. This highly scalable technique is widely used in engineering and system design, but it is difficult to simulate all representative behaviors of a system. Hybrid systems are particularly difficult to analyze, since neighboring states, no matter how close, may exhibit qualitatively different behaviors. Even numerical simulation can be difficult for such systems, in particular if high confidence in the result is required. Conventional techniques from model-based design, such as simulation of corner cases or stochastic simulation, may fail to detect critical behavior and require a prohibitive number of simulation runs to be conclusive. To ensure that no critical behaviors are missed, reachability analysis aims at

accurately and quickly computing a cover of the states of the system that are reachable from a given set of initial states. Reachability can be used to formally show safety and bounded liveness properties. Set-based reachability constructs a cover of of all behaviors by exhaustively computing one-step successor states until a fixed-point is found. This can be seen as a generalization of numerical simulation to sets, plus book-keeping to detect previously visited states. In addition to verifying safety properties, set-based reachability can be used to obtain quantitative information, e.g., bounds on the jitter in an oscillator circuit. Hybrid and cyber-physical systems in a wide range of application domains have been analyzed through reachability analysis, e.g., automotive control [40], robotics [77], electronic circuits [43], and systems biology [30].

A major obstacle to applying reachability analysis in practice is scalability. In general, one-step successors can only be computed approximately and are difficult to scale in the number of continuous variables. The approximation error requires particular attention since it can accumulate rapidly, leading to a coarse cover, prohibitive state explosion, or preventing termination. The complexity of one-step successor computations depends decisively on the dynamics of the system. Piecewise-affine dynamics are particularly suitable, since covering the solution of affine ODEs can be reduced to a linear problem once an approximation for the initial time-step is available [10]. Several different set representations have been proposed for obtaining the cover. In the year 2000, a scalable approach based on ellipsoids was proposed [61, 63]. However, ellipsoidal techniques do not easily extend to hybrid systems because ellipsoids are not closed under essential set operations such as Minkowski sum, convex hull, and intersection. Approximation errors can quickly accumulate. In 2005, a particular type of polytope, called zonotope, was successfully used for reachability analysis of hybrid systems [59, 49]. Zonotopes are a subclass of central-symmetric polytopes that is closed under Minkowski sum, and for which good approximations of the convex hull can be efficiently obtained. While more computationally intensive than ellipsoids, the approximation error can be made smaller. However, zonotopes are not closed under intersection, which can make the computation of jump successors problematic. In special cases, an approach called continuization (interpolation between dynamics of neighboring locations) can help to avoid the intersection operation [3].

In 2006, Le Guernic and Girard achieved a breakthrough with an algorithm for affine dynamics that allowed them to avoid the error accumulation during time elapse computation, known as the wrapping effect [51]. Using zonotopes, precise image computations became possible for systems with hundreds of variables. A second breakthrough was reached in 2008, when Le Guernic and Girard proposed to use support functions as set representations [50]. Support functions had been proposed for approximating the reachable set of linear ODEs in [52, 13, 87] and the Russian literature cited in [68]. If the set of initial states is convex, these approaches define a convex approximation (a polyhedron) for each point in the dense time domain. Over an interval of time, one obtains an uncountable number of polyhedra, whose union is in general not convex. Le Guernic and Girard found a way to efficiently compute an approximation of this union with a single convex set, and then propagate this approximation forward in time to cover the entire reachable set up to a given bounded time horizon. As a result, the reachable states are covered with a finite number of convex sets, each of which can be computed efficiently. The approximation error

is linear in size of the initial time interval. The approach was extended to hybrid systems in [66]. Support functions have the advantage over zonotope techniques that arbitrary compact convex sets can be represented, and that approximations can be computed lazily, with the possibility to incrementally refining the result up to the desired accuracy.

1.1 Contributions

The work presented in this monograph is a collection of theoretical and heuristic improvements of the support function approach by Le Guernic and Girard, with the central aim to obtain a scalable algorithmic approach that makes set-based reachability applicable in practice. The contributions are as summarized in the following paragraphs, following the outline of the chapters. Chapter 2 provides an introduction to reachability and support functions are defined in Sect. 3.1.

Template Reachability: The template reachability approach by Le Guernic and Girard from [66] was complemented by the book-keeping required for a fixed-point reachability algorithm containment checking, and implemented on the SpaceEx platform. The underlying ODE-approximation was improved in accuracy by intersecting approximation bounds going forward and backwards in time; this heuristic lead to accuracy improvements of many orders of magnitude. The fixed-step algorithm was extended to variable time steps, where the time steps are adjusted separately for each template direction. The approach is described in Chapter 3 and published in [44].

One-Step Template Refinement: The accuracy of template reachability depends highly on the a-priori choice of the template. In principle, the support function approach allows one to side-step this problem through nested evaluations of the support functions of predecessor states, all the way back to the initial states. However, the number of evaluations grows exponentially with the depth of nesting, to the point where nesting is impractical. It is possible to approximate the solution set up to a given precision by adding template directions [52], but in general the number of directions grows exponentially with the dimension [68]. For the sake of scalability, we add directions to increase the accuracy only with respect to the original template directions. From the structure of our reachability algorithm, this is best carried out before the intersection with the so-called guard condition, which specifies when a state can jump. We developed a solution for guards that are halfspaces or hyperplanes, and extend the approach heuristically to polyhedral guards. In experiments the increase in precision is rewarded by a faster convergence of the reachability algorithm, leading to an overall approach that is both faster and more precise. The approach is described in Chapter 4 and published in [46].

Piecewise-Linear Flowpipe Approximations and Clustering: In the support function approach, the complexity of successor computations for each individual set is a function of the number of template directions, which in turn determine the accuracy. If one accepts the resulting approximation error, the successor computations are highly scalable. But this does not yet lead to a scalable fixed-point algorithm. Under these premises, it is the number of sets produced by successor computations that constitutes the major bottleneck. In the original

approach [66], increasing the accuracy with respect to a particular direction forcibly increases the number of sets. Heuristic clustering of sets, as in [44], leads to approximation errors that are wildly unpredictable. We developed an approach that allows us to produce for a given directional accuracy the minimal number of semi-template polyhedra, with the minimal number of constraints, covering the reachable states in space-time (the state space plus time as additional dimension). As a key ingredient, the nonconvex flowpipe approximations are represented by sets of piecewise linear functions, on which many geometric operations can be computed very efficiently. To the best of my knowledge, these representations are novel. The approach is described in Chapter 5 and published in [42, 38].

Property-Driven Refinement and Underapproximations of the Support Function: As a second approach to template refinement, we considered the problem of refining template directions just enough to show non-emptiness of a one-step successor computation. This can also be considered as solving the safety problem for a continuous system with affine dynamics, but with a focus on fast resolution rather than completeness. The technique was applied in [39] to eliminate spurious transitions, which are frequently at the heart of the explosive growth in the number of states. We propose underapproximations of the support functions [42], which to my knowledge are novel. These can suffice to show or refute certain geometric properties, like disjointness, before it is possible to deduce a single point that is actually in the set. The approach is described in Chapter 6.

SpaceEx: The software tool SpaceEx was designed and implemented as an experimentation platform for developing and evaluating competing reachability algorithms. Its architecture is generic in the sense that it suits all set-based reachability algorithms known to the author. Implementations of wildly differing algorithms, such as the PHAVer algorithm [35] and the support function approaches from [66] and [42], show the competitive performance. Third-party additions are available from the University of Freiburg [17, 18] and further are being developed by the Technical University of Munich in the EU project UnCoVerCPS [85]. The platform architecture is outlined in Chapter 7 and published in [45].

1.2 Research Supervision and Projects

I had the pleasure and privilege to work with a number of PhD students and PostDocs, with funding from several projects that are briefly described below.

PhD students: From 2008 to 2012 I co-supervised, together with Oded Maler, Rajarshi Ray. Rajarshi is now an assistant professor at the National Institute of Technology, Meghalaya, India. He was instrumental in developing SpaceEx and its architecture as described in Chapter 7, and wrote his thesis on the implementation and improvements of the support function algorithms in Chapters 3 and 4. His work was published in [44, 45, 46]. Since April 2015 I've been co-supervising, together with Oded Maler, Nikos Kektatos. Nikos is currently investigating the combination of trajectory computation with set-based reachability.

PostDoc: Stefano Minopoli has been working with me as a PostDoc since April 2013. He has worked on reachability with urgent semantics, as well as translating simulation models to hybrid automata. Our work was published in [73, 75, 74].

External collaborations: I have collaborated with Xin Chen during his time as PhD student of Erika Ábrahám. This lead to publications on polynomial-complexity reachability for hybrid systems with piecewise constant derivatives [23], benchmarks for hybrid systems [25], and a survey on reachability techniques [81]. Another fruitful collaboration developed with Sergiy Bogomolov during his time as PhD student of Andreas Podelski. The resulting work on high-level improvements and applications of set-based reachability was published in [16, 17, 19, 18]. We further worked on support function reachability in [39]

SpaceEx contributors: I would also like to acknowledge the contributions of people involved in the creation of the SpaceEx platform, amongst others Rodolfo Ripado, Scott Cotton, Rajat Kateja, and Manish Goyal.

The research presented here was partly financed by the following projects, whose support I gratefully acknowledge:

- The FP7 EU project Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systemes (MULTIFORM) started in September 2008 and terminated in February 2012. I was co-founder and PI. The academic partners were the Universities of Dortmund, Eindhoven, Aachen, Aalborg, and Grenoble (UJF - Verimag); the industrial partners were the Stichting Embedded Systems Institute; VEMAC and KVCA.

- The H2020 EU project Unifying Control and Verification of Cyber-Physical Systems (UnCoVerCPS) started in January 2015 and is expected to run until December 2018. I was one of the co-founders et PI. The academic partners in the project are the Universities of Munich (TUM), Kassel, Milano, and Grenoble (UJF/Verimag); and the industrial partners are General Electric, Bosch, Esterel Technologies, DLR, Tecnalia, and R.U.Robots.

- The sub-project Variabilité des circuits analogique hiérarchiques of the project NANO2017 started in January 2015 and runs until december 2017. I was one of the co-founders and PI. The partners are STMicroelectronics and UJF/Verimag.

The development of the SpaceEx platform was supported in part through two one-year projects of the Institute Carnot-LSI, in 2014 and 2015.

Chapter 2

Hybrid Automata and Reachability

Hybrid automata are a modeling formalism that combines discrete states with continuously evolving, real-valued variables. The discrete states and the possible transitions from one state to another are described with a finite state-transition system. A change in discrete state can update the continuous variables and modify the set of differential equations that describes how variables evolve with time. Hybrid automata are non-deterministic, which means that different futures may be available from any given state. Rates of change or variable updates can be described by providing bounds instead of fixed numbers. Incomplete knowledge about initial conditions, perturbations, parameters, etc. can easily be captured this way. Hybrid automata capture a rich variety of behaviors, and are used in a wide range of domains such as automotive control, robotics, electronic circuits, systems biology, and health care. The hybrid automaton model is well-suited for formal analysis, in the sense that sets of behaviors are readily described by mathematical equations.

In Sect. 2.1 we give a formal definition of hybrid automata and their semantics. The fundamentals of set-based reachability are discussed in Sect. 2.2. A specific instance of successor operations for piecewise affine dynamics is described in Sect. 2.3. It builds the foundation for the improvements in Chapters 3 and 5. Related work is indicated throughout the text, but given the rich literature on the topic this introduction is far from exhaustive. For further reading, see [70, 5, 83].

2.1 Hybrid Automata

Hybrid automata describe the evolution of a set of real-valued variables over time. In this section, we give a formal definition of hybrid automata and their behaviors, and illustrate the concept with an example. But first, we introduce some notation for describing real-valued variables, and sets of these values in the form of predicates and polyhedra.

2.1.1 Preliminaries

Variables: Let $X = \{x_1, \ldots, x_n\}$ be a finite set of identifiers we call *variables*. Attributing a real value to each variable we get a *valuation* over X, written as $x \in \mathbb{R}^X$ or $x : X \to \mathbb{R}$. We will use the primed variables $X' = \{x'_1, \ldots, x'_n\}$ to denote successor values and the dotted variables $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ to denote the derivatives of the variables with respect to time. Given a set of variables $Y \subseteq X$, the *projection* $y = x \downarrow_Y$ is a valuation over Y that maps each variable in Y to the same value that it has in x. We may simply use a vector $x \in \mathbb{R}^n$ if it is clear from the context which index of the vector corresponds to which variable. We denote the i-th element of a vector x as x_i or $x(i)$ if the latter is ambiguous. In the following, we use $\mathbb{R}^n$ instead of $\mathbb{R}^X$ except when the correspondance between indices and variables is not obvious, e.g., when valuations over different sets of variables are involved.

Predicates: A *predicate* over X is an expression that, given a valuation x over X, can be evaluated to either true or false. A *linear constraint* is a predicate

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b,$$

where $a_1, \ldots a_n$ and b are real-valued constants, and whose sign may be strict ($<$) or nonstrict ($\leq$). A linear constraint is written in vector notation as

$$a^\mathsf{T} x \leq b,$$

with coefficient vector $a \in \mathbb{R}^n$ and inhomogeneous coefficient $b \in \mathbb{R}$. A *halfspace* $\mathcal{H} \subseteq \mathbb{R}^n$ is the set of points satisfying a linear constraint. A predicate over X defines a continuous set, which is the subset of $\mathbb{R}^X$ on which the predicate evaluates to true.

Polyhedra: A conjunction of finitely many linear constraints defines an $\mathcal{H}$-*polyhedron*, or polyhedron in *constraint form*,

$$\mathcal{P} = \Big\{ x \;\Big|\; \bigwedge\nolimits_{i=1}^{m} a_i^\mathsf{T} x \bowtie_i b_i \Big\}, \text{ with } \bowtie_i \in \{<, \leq\},$$

with *facet normals* $a_i \in \mathbb{R}^n$ and *inhomogeneous coefficients* $b_i \in \mathbb{R}$. In vector-matrix notation, an $\mathcal{H}$-*polyhedron* can be written as

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

An $\mathcal{H}$-polyhedron is a *closed* set if it can be defined using only nonstrict constraints. A bounded polyhedron is called a *polytope*. Note that the constraints defining $\mathcal{P}$ are not necessarily unique. A closed polyhedron $\mathcal{P}$ can be represented in *generator form* by a pair (V, R), where $V \subseteq \mathbb{R}^n$ is a finite set of *vertices*, and $R \subseteq \mathbb{R}^n$ is a finite set of *rays*. They define the $\mathcal{V}$-*polyhedron*

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

which consists of the convex hull of the vertices, extended towards infinity along the directions of the rays. The generator representation can be extended with *closure points* to deal with non-closed polyhedra [12]. An $\mathcal{H}$-polyhedron can be converted to a $\mathcal{V}$-polyhedron and vice versa, but this may increase the complexity exponentially.

2.1.2 Definition and Semantics

We now give a formal definition of a hybrid automaton and its run semantics.

Definition 2.1 (Hybrid automaton)**.** *[6, 57] A* hybrid automaton

$$H = (\mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump})$$

consists of

- *a finite set of* locations $\mathsf{Loc} = \{\ell_1, \ldots, \ell_m\}$ *representing the discrete states,*
- *a finite set of* synchronization labels Lab*, also called its* alphabet*, which can be used to coordinate state changes between several automata,*
- *a finite set of edges* $\mathsf{Edg} \subseteq \mathsf{Loc} \times \mathsf{Lab} \times \mathsf{Loc}$*, also called* transitions*, which determines which discrete state changes are possible using which label,*
- *a finite set of* variables $X = \{x_1, \ldots, x_n\}$*, partitioned into uncontrolled variables U and controlled variables Y; a* state *of H consists of a location ℓ and a value for each of the variables, and is denoted by $s = (\ell, x)$;*
- *a set of states* Inv *called* invariant *or* staying condition*; it restricts for each location the values that x can possibly take and so determines how long the system can remain in the location;*
- *a set of* initial *states* $\mathsf{Init} \subseteq \mathsf{Inv}$*; every behavior of H must start in one of the initial states;*
- *a* flow relation Flow*, where* $\mathsf{Flow}(\ell) \subseteq \mathbb{R}^{\dot{X}} \times \mathbb{R}^X$ *gives for each state (ℓ, x) the set of possible derivatives $\dot{x}$, e.g., using a differential equation such as*

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

  Given a location ℓ, a trajectory *of duration $\delta \geq 0$ is a continuously differentiable function $\xi : [0, \delta] \to \mathbb{R}^X$ such that for all $t \in [0, \delta]$, $(\dot{\xi}(t), \xi(t)) \in \mathsf{Flow}(\ell)$. The trajectory* satisfies the invariant *if for all $t \in [0, \delta]$, $\xi(t) \in \mathsf{Inv}(\ell)$.*
- *a* jump relation Jump*, where* $\mathsf{Jump}(e) \subseteq \mathbb{R}^X \times \mathbb{R}^{X'}$ *defines for each transition $e \in \mathsf{Edg}$ the set of possible successors x' of x; jump relations are typically given by a* guard *set $\mathcal{G} \subseteq \mathbb{R}^X$ and an assignment (or reset) $x' = r(x)$ as*

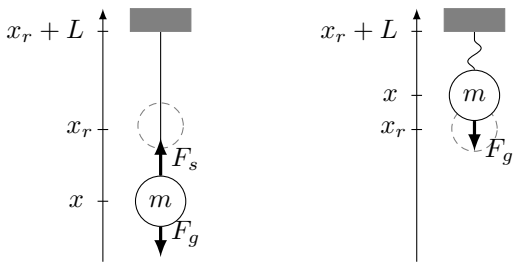
  $$\mathsf{Jump}(e) = \{(x, x') \mid x \in \mathcal{G} \wedge x' = r(x)\}.$$

We define the behavior of a hybrid automaton with a *run*: starting from one of the initial states, the state evolves according to the differential equations whilst time passes, and according to the jump relations when taking an (instantaneous) transition. Special events, which we call *uncontrolled assignments*, model an environment that can make arbitrary changes to the uncontrolled variables.

Definition 2.2 (Run semantics)**.** *A* run *of H is a sequence*

$$(\ell_0, x_0) \xrightarrow{\delta_0, \xi_0} (\ell_0, \xi_0(\delta_0)) \xrightarrow{\alpha_0} (\ell_1, x_1) \xrightarrow{\delta_1, \xi_1} (\ell_1, \xi_1(\delta_1)) \ldots \xrightarrow{\alpha_{N-1}} (\ell_N, x_N),$$

with $\alpha_i \in \mathsf{Lab} \cup \{\tau\}$, satisfying for $i = 0, \ldots, N-1$:



(a) *extension* (b) *freefall*

Figure 2.1: A ball suspended from a ceiling by an elastic string

1. *The first state is an initial state of the automaton, i.e.,* $(\ell_0, x_0) \in \mathsf{Init}$.

2. Trajectories: *In location* ℓ_i, ξ_i *is a trajectory of duration* δ_i *that satisfies the invariant.*

3. Jumps: *If* $\alpha_i \in \mathsf{Lab}$, *there exists a transition* $(\ell_i, \alpha_i, \ell_{i+1}) \in \mathsf{Edg}$ *with jump relation* $\mathsf{Jump}(e)$ *such that* $(\xi_i(\delta_i), x_{i+1}) \in \mathsf{Jump}(e)$ *and* $x_{i+1} \in \mathsf{Inv}(\ell_{i+1})$.

4. Uncontrolled assignments: *If* $\alpha_i = \tau$, *then* $\ell_i = \ell_{i+1}$ *and* $\xi_i(\delta_i) \downarrow_Y = x_{i+1} \downarrow_Y$. *This represents arbitrary assignments that the environment might perform on the uncontrolled variables* $U = X \setminus Y$.

A state (ℓ, x) *is* reachable *if there exists a run with* $(\ell_i, x_i) = (\ell, x)$ *for some* i.

Note that the strict alternation of trajectories and jumps in Def. 2.2 is of no particular importance. Two consecutive jumps can be represented by inserting a trajectory with duration zero (which always exists), and two consecutive trajectories can be represented by inserting an uncontrolled assignment jump that does not modify the variables.

May and Must semantics: In Def. 2.2, transitions may be taken when they are enabled, but there is no obligation to do so – the system may remain in a location as long as the invariant is satisfied. These so-called *may* semantics allow one to include nondeterminism about when a transition will be taken, e.g., when it is not clear how fast a discrete controller will react to a stimulus. In the Ball/String example, this could be used if the length of the string (position of the ceiling) is not exactly known. In contrast, *must* or *ASAP* semantics dictate that the transition is taken as soon as possible. These semantics are used by simulators such as Simulink [72], Dymola [20], MapleSim [71], etc., since they require deterministic models. Some verification tools, like HyTech [55] and PHAVer [35], allow one to include both types of transitions.

Example 2.3 (Ball/String)**.** *Consider a ball that is suspended from a ceiling by a string, as shown in Fig. 2.1. We will construct a simple model that only takes into account the vertical movement of the ball. The Ball on String example is modeled by the hybrid automaton with*

- *locations* $\mathsf{Loc} = \{\text{freefall}, \text{extension}\}$,
- *labels* $\mathsf{Lab} = \{\text{up}, \text{down}, \text{bounce}\}$,

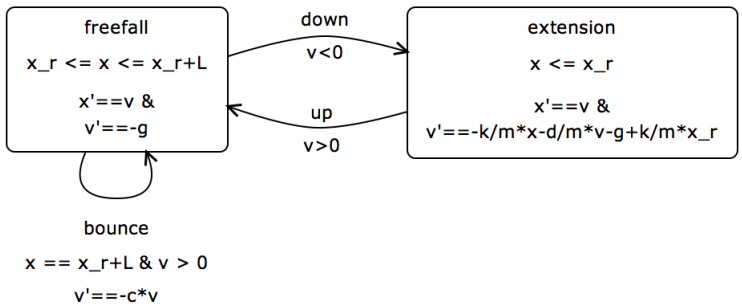
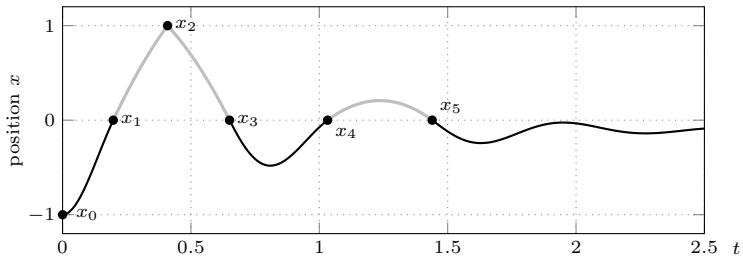


Figure 2.2: A hybrid automaton model of a ball on string, constructed in the tool SpaceEx. In the flow equations, x' denotes the derivative $\dot{x}$

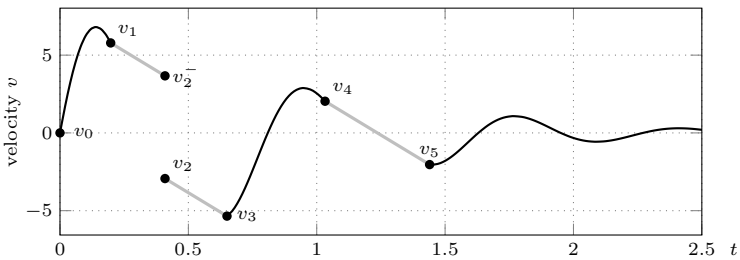
- *transitions* $\mathsf{Edg} = \big\{(\text{freefall}, \text{down}, \text{extension}),\ (\text{extension}, \text{up}, \text{freefall}),\ (\text{freefall}, \text{bounce}, \text{freefall})\big\}$,
- *variables* $X = \{x, v\}$, *with controlled variables* $Y = X$ *and uncontrolled variables* $U = \emptyset$,
- *initial states* $\mathsf{Init} = \{\text{extension}\} \times \{x = -1, v = 0\}$,

and the invariants, flow relations, and jump relations as shown in Fig. 2.2. Since we do not expect x or v to be modified by the environment, we consider both variables to be controlled (as is usually the case for variables whose derivative is given). The transition from freefall *to* extension *has the constraint $v < 0$ since it is only necessary to change the location if the ball is actually moving. Similarly, the transition from* freefall *to* extension *has the constraint $v > 0$. In terms of the behavior of the hybrid automaton for single trajectory, both constraints are redundant. We introduce them here to avoid unnecessary chattering between locations, since it could degrade the accuracy and performance of the reachability analysis that will be presented later.*

The behaviors of the hybrid automaton are derived as follows. In location extension*, the dynamics are those of a damped oscillator. The ODE system is linear in the variables x and v, so its solution is a combination of exponential, sine and cosine functions of time. In location* freefall*, the dynamics are also linear, but of a particularly simple kind. The derivative of v is constant, $\dot{v} = -g$, so that v evolves in a straight line and x in a parabola. Figure 2.4 shows the trajectories of the same run as in Fig. 2.3, but in the state space (also called phase space), which allows one to graph over an infinite time horizon.*

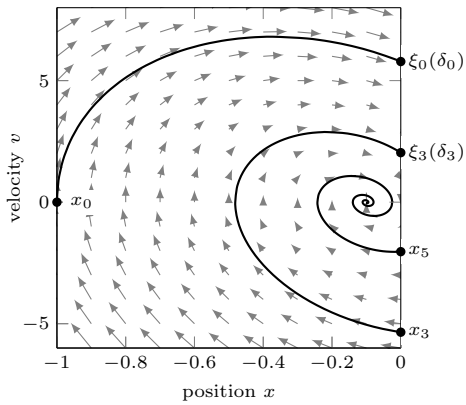


(a) position x over time t

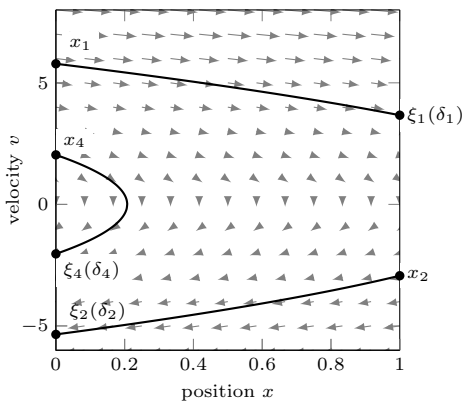


(b) velocity v over time t

Figure 2.3: A run of the ball on string model consists of a sequence of trajectories. Trajectories in location *extension* are shown in black, trajectories in *freefall* in gray



(a) location *extension*



(b) location *freefall*

Figure 2.4: A sample run of the ball/string example, with trajectories $\xi_0, \ldots, \xi_5$. The initial state x_0 corresponds to the variable values $x = -1, v = 0$ in location *extension*. The arrows indicate the direction and magnitude of the derivative

2.2 Set-based Reachability

A 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 state in S,

$$\mathsf{post}_C(S) = \{(\ell, \xi(\delta)) \mid \exists(\ell, x) \in S : (\ell, x) \xrightarrow{\delta,\xi} (\ell, \xi(\delta))\}.$$

Let $\mathsf{post}_D(S)$ be the set of states resulting from a jump from any state in S,

$$\mathsf{post}_D(S) = \{(\ell', x') \mid \exists(\ell', x') \in S, \exists\alpha \in \mathsf{Lab} \cup \{\tau\} : (\ell, x) \xrightarrow{\alpha} (\ell', x')\}.$$

Starting from the initial states, $\mathsf{post}_C(S)$ and $\mathsf{post}_D(S)$ are computed in alternation. All states that are obtained are recorded, as in the following sequence:

$$\begin{array}{rcl} R_0 & = & \mathsf{post}_C(\mathsf{Init}), \\ R_{i+1} & = & R_i \cup \mathsf{post}_C(\mathsf{post}_D(R_i)). \end{array} \tag{2.1}$$

If the sequence reaches a fixed-point, i.e., when $R_{i+1} = R_i$, then R_i is the set of reachable states. Note that simply computing the sequence and testing for a fixed-point may not 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.

In tools such as HyTech [55], PHAVer [35] and SpaceEx [44], the sequence (2.1) is computed using *symbolic states* $s = (\ell, \mathcal{P})$, where $\ell \in \mathsf{Loc}$ and $\mathcal{P}$ is a continuous set, e.g., a polyhedron. Computing the timed successors post_C of a symbolic state $s = (\ell, \mathcal{P})$ produces a new symbolic state $s' = (\ell, \mathcal{P}')$. Computing the jump successors post_D of $s = (\ell, \mathcal{P})$ involves iterating over all outgoing transitions of ℓ, and produces a set of symbolic states $\{s'_1, \ldots, s'_N\}$, each in one of the target locations. A *waiting list* contains the symbolic states whose successors still need to be explored, and a *passed list* contains all symbolic states computed so far. The fixed-point computation proceeds as follows:

1. Initialization: Compute the continuous successors of the initial states and put them on the waiting list.

2. Pop a symbolic state s from the waiting list and compute its one-step successors $\{s'_1, \ldots, s'_N\} = \mathsf{post}_C(\mathsf{post}_D(s))$.

3. Containment checking: Discard the s'_i that have previously been encountered, i.e., those contained in any symbolic state on the passed list. Add the remaining symbolic states to the passed and waiting list.

4. If the waiting list is empty, terminate and return the passed list as the reachable states. Otherwise, continue with step 2.

Different approaches are taken for computing the one-step successors, depending on the type of dynamics. In the following sections, we present the major methods.

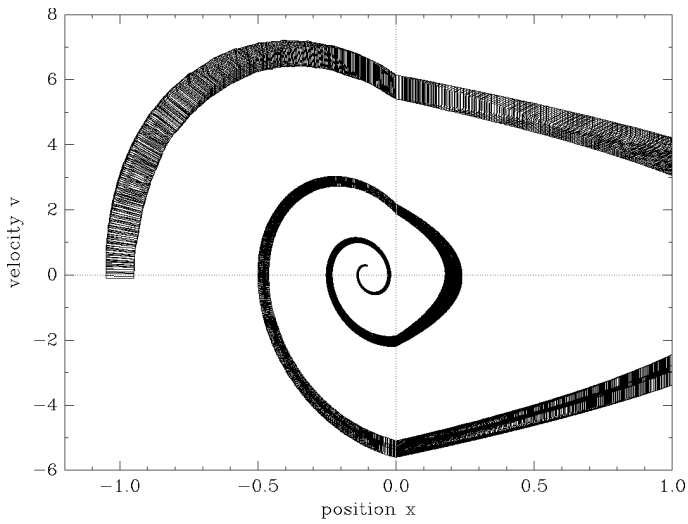


Figure 2.5: Reachable states of the ball/string example, computed using SpaceEx

Example 2.4. *Figure 2.5 shows the reachable states of the ball/string example, starting from an initial set of* $-1.05 \leq x \leq -0.95$, $-0.1 \leq v \leq 0.1$ *in location extension. Initializing the waiting list with the continuous successors of the intial states, the fixed point is reached on the 6th iteration. Each symbolic state corresponds to a segment of the run from Ex. 2.3, and contains all of the corresponding trajectories, from any of the initial states.*

2.3 Reachability for Piecewise Affine Dynamics

Hybrid automata with piecewise affine dynamics (PWA) have

- initial states and invariants given by conjunctions of linear constraints,
- flows given by affine ODEs, and
- jumps given by a guard set and linear assignments.

We divide the continuous variables into *state variables* $X = \{x_1, \ldots, x_n\}$, whose derivative is explicitly defined, and *input variables* $U = \{u_1, \ldots, u_m\}$, 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 given by *affine ODEs* of the form

$$\dot{x} = Ax + Bu, \qquad u \in \mathcal{U}, \tag{2.2}$$

where A and B are matrices of appropriate dimension and the *input set* $\mathcal{U}$ is compact and convex. Note that $\mathcal{U}$ may be specified in the invariant. To simplify notation in this section, we assume that constants are modeled with $\mathcal{U}$, e.g., $\dot{x} = Ax + b$ with $B = I$ and $\mathcal{U} = \{b\}$. Some differential inclusions can be brought to the form of (2.2) by introducing auxiliary variables. The jump

constraints of an edge e are defined by a *guard* set $\mathcal{G}$ and an assignment of the form

$$x' = Cx + Du, \tag{2.3}$$

where x' denotes the value of x after the jump, u is defined as above and C and D are matrices of appropriate dimension.

2.3.1 Continuous successors

We start with a basic version of approximating the successor states reachable by time elapsing. This version ignores the invariant. 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 (2.2) for initial condition $\xi(0) = x_0$ and a given input signal ζ. It has the form

$$\xi_{x_0,\zeta}(t) = e^{At}x_0 + \int_0^t e^{A(t-s)}B\zeta(s)ds. \tag{2.4}$$

It consists of the superposition of the solution of the *autonomous* system, obtained for $\zeta(t) = 0$, and the *input integral* obtained for $x_0 = 0$. 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 (2.4) can be written as

$$\mathcal{X}_t = e^{At}\mathcal{X}_0 \oplus \mathcal{Y}_t. \tag{2.5}$$

The goal is to compute a finite sequence of sets $\Omega_0, \Omega_1, \ldots$ such that

$$\bigcup_{0\leq t\leq T} \mathcal{X}_t \subseteq \Omega_0 \cup \Omega_1 \cup \ldots. \tag{2.6}$$

We present the construction of the sequence Ω_k for a fixed *time step* $\delta > 0$ such that Ω_k covers $\mathcal{X}_t$ for $t \in [k\delta, (k+1)\delta]$, as illustrated in Fig. 2.6. The so-called *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 (2.5), we obtain that for any $r, s \geq 0$,

$$\mathcal{X}_{r+s} = e^{Ar}\mathcal{X}_s \oplus \mathcal{Y}_r. \tag{2.7}$$

Substituting $r \leftarrow \delta$, $s \leftarrow k\delta$, we get a time discretization

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

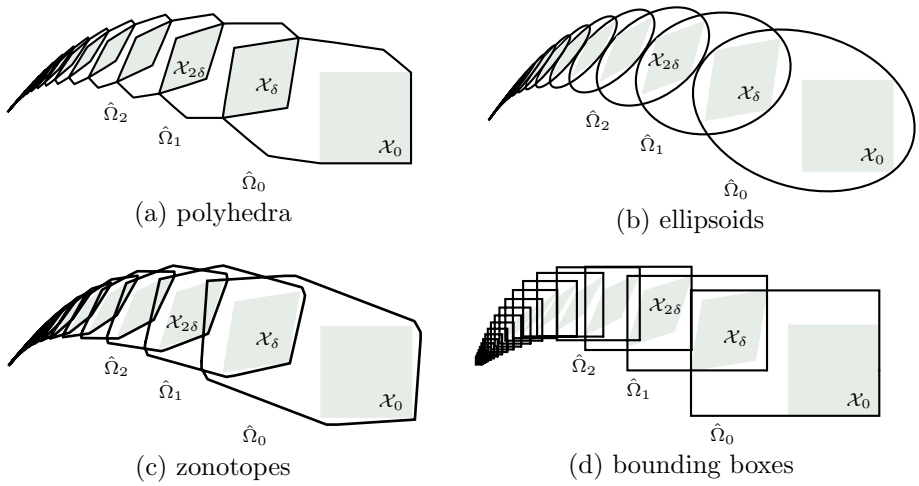
It follows that if we have initial approximations Ω_0 and Ψ_δ such that

$$\bigcup_{0\leq t\leq \delta} \mathcal{X}_t \subseteq \Omega_0, \qquad \mathcal{Y}_\delta \subseteq \Psi_\delta, \tag{2.8}$$

then the sequence

$$\Omega_{k+1} = e^{A\delta}\Omega_k \oplus \Psi_\delta. \tag{2.9}$$

satisfies (2.6). 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.



(a) polyhedra (b) ellipsoids

(c) zonotopes (d) bounding boxes

Figure 2.6: A sequence of sets $\Omega_0, \Omega_1, \ldots$ (solid outline) that covers $\mathcal{X}_t$ (shaded light green) over a finite time horizon T. The choice of set representation for Ω_k has a substantial impact on accuracy and computational complexity

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. 2.7(a), is to compute the convex hull in constraint representation, and push the facets out far enough to be conservative [53]. The required values can be computed from a Taylor approximation of (2.4) [10], or by solving an optimization problem [26]. 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, as illustrated in Fig. 2.7(b). The approach from [49] 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 $\mathcal{X}_\delta$ [65]. The bloating factor is derived from a Taylor approximation of (2.4), 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.[1] 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 $c\mathcal{X} = \{cx \mid x \in \mathcal{X}\}$.

Lemma 2.5. *[49] Given a set of initial states $\mathcal{X}_0$ and affine dynamics (2.2), let*

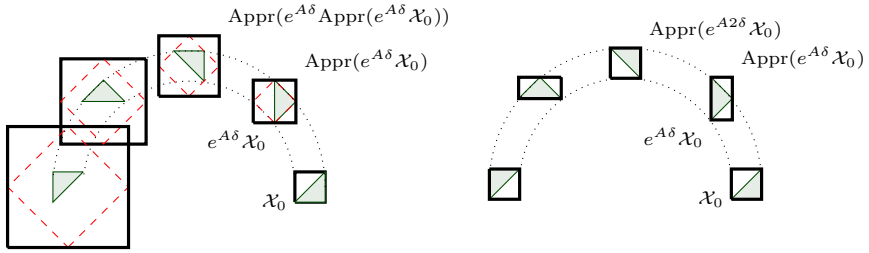
$$\begin{aligned}
\alpha_\delta &= \mu(\mathcal{X}_0) \cdot (e^{\|A\|\delta} - 1 - \|A\|\delta),\\
\beta_\delta &= \tfrac{1}{\|A\|}\mu(B\mathcal{U}) \cdot (e^{\|A\|\delta} - 1),\\
\Omega_0 &= \mathrm{CH}(\mathcal{X}_0 \cup e^{A\delta}\mathcal{X}_0) \oplus (\alpha_\delta + \beta_\delta)\mathcal{B},\\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

Then $\bigcup_{0\le t\le\delta} \mathcal{X}_t \subseteq \Omega_0$ and $\mathcal{Y}_\delta \subseteq \Psi_\delta$.

[1] For example, the *infinity norm* $\|x\|_\infty = \max\{|x_1|, \ldots, |x_n|\}$ 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.

(a) pushing facets (b) bloating

Figure 2.7: 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 compensating for the curvature of trajectories



(a) wrapping effect (b) wrapping-free algorithm

Figure 2.8: An example for the wrapping effect, with $e^{A\delta}$ performing a rotation of 45 degrees around the origin. The exact solution is $e^{Ak\delta}\mathcal{X}_0$ (shaded). Mapping (dashed) and then applying the approximation operator (thick) at each step leads to the wrapping effect. For visual clarity, $\mathcal{X}_0$ is used here instead of Ω_0

Approximations and the wrapping effect: The sequence in (2.9) can be problematic to compute since the complexity of Ω_k may increase sharply with 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 $\mathcal{P}$, $\mathcal{P} \subseteq \mathrm{Appr}(\mathcal{P})$. The sequence (2.9) then becomes

$$\hat{\Omega}_{k+1} = \mathrm{Appr}(e^{A\delta}\hat{\Omega}_k \oplus \Psi_\delta). \tag{2.10}$$

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* [59] and is illustrated in Fig. 2.8.

For affine dynamics, the wrapping effect can be avoided by combining two techniques [51]. First, the alternation of the map $e^{Ak\delta}$ with the Minkowski sum in (2.9) is avoided by splitting it into two sequences

$$\begin{aligned} \hat{\Psi}_{k+1} &= \mathrm{Appr}(e^{Ak\delta}\Psi_\delta) \oplus \hat{\Psi}_k, \qquad \text{with } \hat{\Psi}_0 = \{0\}, \\ \hat{\Omega}_k &= \mathrm{Appr}(e^{Ak\delta}\Omega_0) \oplus \hat{\Psi}_k. \end{aligned} \tag{2.11}$$

Second, the approximation operator is chosen such that

$$\mathrm{Appr}(\mathcal{P} \oplus \mathcal{Q}) = \mathrm{Appr}(\mathcal{P}) \oplus \mathrm{Appr}(\mathcal{Q}),$$

which is the case, e.g., for the interval hull (bounding box). Under this assumption it holds that $\hat{\Omega}_k = \mathrm{Appr}(\Omega_k)$, which means the resulting approximation is free of the wrapping effect.

Invariants: A simple but frequently sufficient heuristic to account for the invariant is to stop computing the sequence Ω_k as soon as Ω_k lies completely outside of the invariant. The computed $\Omega_0, \ldots, \Omega_{k-1}$ are then intersected with $\mathsf{Inv}(\ell)$, which produces an overapproximation of the exact solution. A more precise solution can be obtained by intersecting at each step with the set of states reachable from the invariant itself [66].

2.3.2 Discrete successors

The *discrete successors* of a polyhedron $\mathcal{P}$ for an edge $e = (\ell, \alpha, k)$ is the polyhedron:

$$\mathsf{post}_e(P)\big\{x' \mid \exists x \in \mathcal{P} : (x, x') \in \mathsf{Jump}(\epsilon) \wedge x' \in \mathsf{Inv}(k)\big\}.$$

This set is defined using existential quantification, and computing it may require costly quantifier elimination. Frequently occurring special cases can be computed more efficiently. Consider an edge $e = (\ell, \sigma, k)$ of a PWA, with guard set $\mathcal{G}$ and assignment

$$x' = Cx + Du,$$

with constant matrices C, D of appropriate dimensions Recall that $u \in \mathcal{U}$, where $\mathcal{U}$ is compact, convex and given by constraints in $Inv(\ell)$. The discrete successors of a set $\mathcal{P}$ is

$$\mathsf{post}_e(\mathcal{P}) = \big(C(\mathcal{P} \cap \mathcal{G}) \oplus D\mathcal{U}\big) \cap \mathsf{Inv}(k).$$

Now consider the assignment to be deterministic, i.e., $x' = Cx + d$, with a constant vector d of appropriate dimension. If C is invertible and $\mathcal{P}, \mathcal{G}$ are $\mathcal{H}$-polyhedra, the computation is straightforward since intersection corresponds to concatenation of constraints, and for any polyhedron $\mathcal{Q} = \{x \mid Ax \leq b\}$,

$$C\mathcal{Q} \oplus \{d\} = \{x \mid AC^{-1}x \leq b + C^{-1}d\}.$$

2.3.3 Set Representations

Whether the presented successor operators $\mathsf{post}_\ell(\mathcal{P})$ and $\mathsf{post}_e(\mathcal{P})$ are efficient to compute, depends on the type of set used for $\mathcal{P}$ and how it is represented. We summarize some of the set representations proposed in literature. Scalable implementations and approximations need to be available for the operators in the algorithm. Using the initial approximation from Lemma 2.5 and the recurrence equation (2.11), the operators are linear map, Minkowski sum, convex hull and intersection.

Polyhedra: Figure 2.6(a) shows a reach set approximation computed using 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. E.g., intersection is computed on $\mathcal{H}$-polyhedra and Minkowski sum on $\mathcal{V}$-polyhedra, and the result can be of exponential complexity in both forms. A polyhedral approximation for the non-scalable operations can be efficiently

computed by a-priori fixing the facet normals of the result, which leads to so-called *template polyhedra*. The accuracy of the approximation can be increased by including additional directions, leading to a scalable approach [11].

Ellipsoids: A scalable reachability algorithm for affine dynamics is obtained for ellipsoids [63], see Fig. 2.6(b). An *ellipsoid* $\mathcal{E}(c, Q) \subseteq \mathbb{R}^n$ is represented by a center $c \in \mathbb{R}^n$ and a positive definite[2] matrix $Q \in \mathbb{R}^{n \times n}$,

$$\mathcal{E}(c, Q) = \big\{x \mid (x - c)^{\mathsf{T}} Q^{-1} (x - c) \leq 1\big\}.$$

Deterministic affine transforms can be computed efficiently for ellipsoids. However, ellipsoids are not closed under Minkowski sum, convex hull, nor intersection. The algorithm in Sect. 2.3.1 therefore suffers from the wrapping effect when implemented with ellipsoids, unless $B\mathcal{U}$ is a singleton. The wrapping effect can be avoided by using approximations over continuous rather than discrete time, as in [87, 63]. 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 [64].

Zonotopes: Zonotopes are a subclass of central-symmetric polytopes that has been used successfully for reachability analysis [49, 4], see Fig. 2.6(c). A *zonotope* $\mathcal{P} \subseteq \mathbb{R}^n$ is defined by a center $c \in \mathbb{R}^n$ and generators $v_1, \ldots, v_k \in \mathbb{R}^n$ as

$$\mathcal{P} = \big\{c + \sum\nolimits_{i=1}^{k} \alpha_i v_i \mid \alpha_i \in [-1, 1]\big\}.$$

Affine transformations and Minkowski sum can be computed efficiently for zonotopes. 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 (2.11). Zonotopes are neither closed under convex hull, nor under intersection. But 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 [3].

2.3.4 Clustering

The accuracy of the approximation in Lemma 2.5 depends on the size of the time step. This property, common to all approaches cited in Sect. 2.3, 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 iteration of the fixed point computation, each one of these sets may become the initial set of yet another sequence, easily leading to an exponential increase in the number of sets.

If only very few of these sets intersect with the guard sets, the discrete successor computation 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

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

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 [66]. 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 or predict. An approach to obtain an optimal number of clusters for a given error bound is presented in [42].

2.3.5 Extension to Nonlinear Dynamics

We describe an approach to deal with nonlinear dynamics

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

where f is usually assumed to be globally Lipschitz continuous. These dynamics can be approximated piecewise with affine dynamics $\dot{x} = Ax + u, u \in \mathcal{U}$. Reachability algorithms for piecewise affine dynamics can then be applied to the approximation. The approximation is nondeterministic, so that all solutions of the original dynamics are covered.

First, the states are confined to a bounded domain $\mathcal{S}$. This could be the invariant in a location, or $\mathcal{S}$ can be derived from 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 matrix elements

$$a_{ij} = \frac{\partial f_i}{\partial x_j}\Big|_{x=x_0} \text{ and} b = f(x_0) - Ax_0.$$

Finally, one derives a set $\mathcal{U}_\epsilon$ that bounds the error such that for all $x \in \mathcal{S}$,

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

Such bounds can be obtained using, e.g., interval arithmetic or optimization techniques. The states reachable using the affine dynamics

$$\dot{x} = Ax + u,\ u \in \mathcal{U}_\epsilon \oplus \{b\}$$

cover those of the original nonlinear dynamics. The accuracy of the linearization depends on the size of the domain $\mathcal{S}$ and can be increased by partitioning $\mathcal{S}$ into smaller parts. This process is known as *phase portrait approximation* [56]. It can be useful even when dealing with purely continuous dynamical systems, and is also known as *hybridization* [8].

Chapter 3

Template Reachability with Support Functions

In this chapter, we describe an efficient and scalable reachability algorithm, which builds on the one in [67] and is adapted specifically for maximum scalability. We present its extension to variable time steps, and propose an improved approximation model, which drastically improves the accuracy of the algorithm. Experimental results demonstrate the scalability of the algorithm and the performance of its implementation in the SpaceEx tool, which is described in more detail in chapter 7.

The chapter is structured as follows. Section 3.1 presents support functions, which are used to represent convex sets and derive the scalable geometric operations. Section 3.2 describes the variable time step algorithm and the new approximation model used to compute time elapse successor states. The computation of successor states of discrete transitions is presented in Sect. 3.3. Experimental results based on our implementation in SpaceEx are provided in Sect. 3.5.

3.1 Support Functions and Template Polyhedra

A convex set can be represented by its support function, which attributes to each direction in $\mathbb{R}^n$ the signed distance of the farthest point of the set to the origin. Computing the value of the support function for a given set of directions, one obtains a polyhedron that overapproximates the set. In this paper, we consider this computation to be approximative, i.e., only a lower and an upper bound on the support function can be computed.

Support Functions The *support function* of a closed and bounded continuous set $\mathcal{X} \subseteq \mathbb{R}^n$ with respect to a direction vector $\ell \in \mathbb{R}^n$ is [15]

$$\rho_{\mathcal{X}}(\ell) = \max\{\ell^{\mathsf{T}} x \mid x \in \mathcal{X}\}.$$

Figure 3.1(a) shows an illustration of the geometrical interpretation of the support function, which is the position of a tangent halfspace that is tangent to

(a) support function in direction ℓ (b) outer approximation

Figure 3.1: Evaluating the support function in a set of directions gives a polyhedral outer approximations that can be computed very efficiently

and contains the set. The support function of a convex set $\mathcal{X}$ is an exact representation of the set:

$$\mathcal{X} = \bigcap_{\ell \in \mathbb{R}^n} \{x \mid \ell^\mathsf{T} x \leq \rho_{\mathcal{X}}(\ell)\}.$$

A point $x^* \in \mathcal{X}$ is called a *support vector* or *maximizer* of $\mathcal{X}$ in direction ℓ if

$$\ell^\mathsf{T} x^* = \rho_{\mathcal{X}}(\ell).$$

The set of maximizers for ℓ is denoted by $\sigma_{\mathcal{X}}(\ell)$. The set of *support vectors* (or *maximizers*) of $\mathcal{X}$ in direction ℓ is denoted by

$$\sigma_{\mathcal{X}}(\ell) = \{x^* \in \mathcal{X} \mid \ell^\mathsf{T} x^* = \rho_{\mathcal{X}}(\ell)\}.$$

Geometric Operations We are interested in support functions because many set operations are computationally cheaper to carry out on support functions than on, say polyhedra [66, 48]. We consider the following algebraic operations on sets $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{R}^n$. The *linear map* $M\mathcal{X} \subseteq \mathbb{R}^n$ with matrix $M \in \mathbb{R}^{n \times n}$ is

$$M\mathcal{X} = \big\{Mx \;\big|\; x \in \mathcal{X}\big\}.$$

The *Minkowski sum* $\mathcal{X} \oplus \mathcal{Y} \subseteq \mathbb{R}^n$ is

$$\mathcal{X} \oplus \mathcal{Y} = \big\{x + y \;\big|\; x \in \mathcal{X}, y \in \mathcal{Y}\big\}.$$

The *convex hull* $\mathrm{CH}(\mathcal{X}) \subseteq \mathbb{R}^n$ of a set $\mathcal{X}$ is

$$\mathrm{CH}(\mathcal{X}) = \Big\{\sum_{i=1}^{m} \lambda_i v_i \;\Big|\; v_i \in X, \lambda_i \in \mathbb{R}^{\geq 0}, \sum_{i=1}^{m} \lambda_i = 1\Big\}.$$

These operations can be very expensive for polyhedra in constraint representation [84], while for support functions they are simple:

$$\rho_{M\mathcal{X}}(\ell) = \rho_{\mathcal{X}}(M^\mathsf{T}\ell), \tag{3.1}$$

$$\rho_{\mathcal{X}_1 \oplus \mathcal{X}_2}(\ell) = \rho_{\mathcal{X}_1}(\ell) + \rho_{\mathcal{X}_2}(\ell), \tag{3.2}$$

$$\rho_{\mathrm{CH}(\mathcal{X}_1 \cup \mathcal{X}_2)}(\ell) = \max\left(\rho_{\mathcal{X}_1}(\ell), \rho_{\mathcal{X}_2}(\ell)\right). \tag{3.3}$$

Template Polyhedra Our reachability algorithm requires two operations for which support functions are not efficient: intersection and deciding containment. For these operations, we use another set representation, *template polyhedra*, which are polyhedra with faces whose normal vectors are given a priori. Given a set $D = \{\ell_1, \ldots, \ell_m\}$ of vectors in $\mathbb{R}^n$ called *template directions*, a template polyhedron $\mathcal{P}_D \subseteq \mathbb{R}^n$ is a polyhedron for which there exist coefficients $b_1, \ldots, b_m \in \mathbb{R}$ such that

$$\mathcal{P}_D = \Big\{ x \in \mathbb{R}^n \;\Big|\; \bigwedge\nolimits_{\ell_i \in D} \ell_i \cdot x \leq b_i \Big\}.$$

Template polyhedron lead to a particularly compact representations when working with large sets of template polyhedra, since the template directions need to be stored only once for the whole set. In order to go from a support function representation of a set $\mathcal{S}$ to a template polyhedron, we use its *template hull* or *outer polyhedron*, which is the template polyhedron defined by coefficients $b_i = \rho(\ell_i, \mathcal{S})$,

$$\lceil \mathcal{S} \rceil_D = \Big\{ x \in \mathbb{R}^n \;\Big|\; \bigwedge\nolimits_{\ell_i \in D} \ell_i \cdot x \leq \rho(\ell_i, \mathcal{S}) \Big\}. \tag{3.4}$$

Figure 3.1(b) shows an illustration. The support function of a polyhedron can be computed efficiently for a given direction ℓ using linear programming. We consider in particular the following sets of template directions:

- $2n$ *box* directions: $x_i = \pm 1$, $x_k = 0$ for $k \neq i$;
- $2n^2$ *octagonal* directions: $x_i = \pm 1$, $x_j = \pm 1$, $x_k = 0$ for $k \neq i$ and $k \neq j$;
- m *uniform* directions (as evenly as possible distributed over the unit sphere).

However, our algorithms support a more general choice of directions, which remains to be investigated. The use of both support functions and template hulls is justified by the fact that they are efficient for different operations, and both set representations are present in our implementation. Support functions are an exact and complete representation of convex sets – implemented as function objects, they can compute values for any direction. With template hulls, the directions D are fixed once and for all at the time of construction, and information for other directions is lost. By switching representations only when necessary (data-dependently) we remain as precise as possible.

3.2 Flowpipe Approximation

We consider the affine continuous dynamics

$$\dot{x}(t) = Ax(t) + u(t), \qquad u(t) \in \mathcal{U}, \tag{3.5}$$

Following the flowpipe approximation approach from Sect. 2.3, we compute a *flowpipe*, a sequence of continuous sets $\Omega_0, \ldots, \Omega_{N-1}$ that covers the reachable states up to a given time horizon T (N depends on the chosen time steps). Before we present the actual algorithm, we discuss how we take into account the invariant of the corresponding location of the hybrid automaton. In our implementation, we test at the k-th step whether Ω_k is entirely outside of the

invariant, and stop the sequence once this is the case. Then we intersect the invariant with the computed sequence $\Omega_0, \ldots, \Omega_k$ (see Sect. 3.3 for a discussion of the intersection operation). The resulting sets are an overapproximation of the reachable states, and the overapproximation may be particularly conservative if it includes trajectories that should have been cut off by the invariant in intermediate time steps. A variation of our algorithm with proper handling of invariants is presented in [66]. In practice we have not yet encountered any case where this ad-hoc handling of the invariant results in an excessive overapproximation, but this issue deserves to be investigated further. We typically include the invariant facet normals in the template directions (SpaceEx does this automatically), and find that the result is usually of satisfactory precision.

We use a variation of the approximation sequence (2.9) with variable time steps. Given arbitrary time steps $\delta_0, \delta_1, \ldots$, we construct the sequence Ω_k that covers the set of reachable states. Each set Ω_k covers the reachable states in the time interval $[t_k, t_{k+1}]$, where $t_k = \sum_{i=0}^{k-1} \delta_i$. The algorithm is based on two functions $\Omega_m(0, \delta)$ and $\Psi_m(\delta)$, which we call *approximation models*. Given any time interval $[0, \delta]$, the approximation model must produce a set that contains the reachable states, similar to the requirement in the case of a fixed time step (2.8). Suitable implementations of such approximation models could be obtained using a variety of methods, such as interval arithmetic, or using set operations such as those in Lemma 2.5. What we require for our algorithm to be sound is that the approximation models cover the actual reachable states:

$$\mathcal{X}_{0,\delta} \subseteq \Omega_m(0, \delta), \qquad \mathcal{Y}_\delta \subseteq \Psi_m(\delta). \tag{3.6}$$

Each Ω_k is constructed by computing $\Omega_m(0, \delta_k)$, which covers $\mathcal{X}_{0,\delta_k}$, and then shifting this set forward in time so that it covers $\mathcal{X}_{t_k, t_k+\delta_k}$. We compute sequences Ψ_k, Ω_k as follows, with $\Psi_0 = \{0\}$:

$$\begin{aligned} \Psi_{k+1} &= \Psi_k \oplus e^{At_k} \Psi_m(\delta_k), \\ \Omega_k &= e^{At_k} \Omega_m(0, \delta_k) \oplus \Psi_k \end{aligned} \tag{3.7}$$

We can show that $\mathcal{Y}_{t_k} \subseteq \Psi_k$ and that $\mathcal{X}_{t_k, t_k+\delta_k} \subseteq \Omega_k$, see [44], so the sequence Ω_k indeed covers the flowpipe. Representing the sets Ψ_k and Ω_k by their support function, the operators used in (3.7) – linear map and Minkowski sum – can be computed efficiently as discussed in Sect. 3.1.

In the next section, we present the new approximation model which we use to compute Ω_k as in (3.7). In Sect 3.2.2, we provide direction-wise error bounds on this computation, and discuss how to adapt the time steps in order to guarantee given error bounds.

3.2.1 Approximation Model

The approximation quality of the sequence Ω_k evidently hinges on the quality of the approximation model. In [67], an approximation model was proposed that uses the norms of A, $\mathcal{X}_0$ and $\mathcal{U}$ to bound the error of a first-order approximation of the solution of the state equations, see Fig. 3.2 for an illustration. If this allows one to establish an asymptotically optimal error over a given time interval, it is sometimes overly conservative in practice. In [2], the approximation model was improved by using element-wise absolute values instead of matrix norms, which

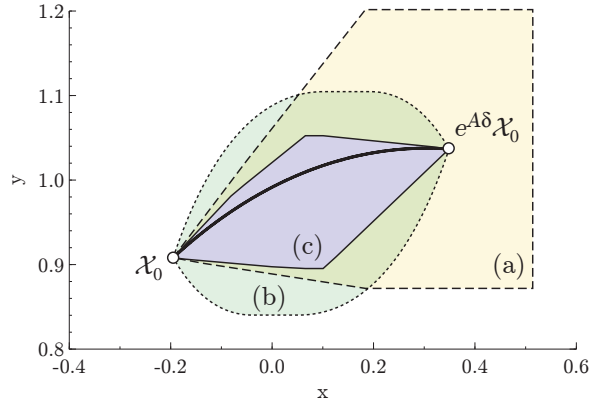


Figure 3.2: Different approximation models for a segment of a circular trajectory, computed by SpaceEx: (a) using a first-order approximation of the ODE [67], (b) using a first-order approximation of the error of the linear interpolation between the states at time $t = 0$ and at $t = \delta$ [65], (c) the new model, which intersects a first-order approximation of the interpolation error going forward in time from $t = 0$ with one that goes backward from $t = \delta$.

can reduce the exponent of error bounds by orders of magnitude. The model we use in this section is a strict improvement over the method in [67] if the norm used is the infinity norm. For other norms, it is possible to find cases for which the resulting sets are incomparable. Similar to the models in [65, 2], it is based on a first-order approximation of the interpolation error between the reachable set at time 0 and at time δ. On top of that, it combines an approximation that goes forward in time with one that goes backward in time in order to further improve the accuracy.

Before presenting the model, we introduce the following notation. The *symmetric interval hull* of a set S, denoted $\boxdot(S)$, is $\boxdot(S) = [-\overline{|x_1|}; \overline{|x_1|}] \times \ldots \times [-\overline{|x_d|}; \overline{|x_d|}]$ where for all i, $\overline{|x_i|} = \sup\{|x_i| \mid x \in S\}$. Let $M = (m_{i,j})$ be a matrix, and $v = (v_i)$ a vector. We define as $|M|$ and $|v|$ the absolute values of M and v respectively, i.e., $|M| = (|m_{i,j}|)$ and $|v| = (|v_i|)$. These absolute values allow us to bound matrix-vector operations (component wise) without taking their norm. The approximation model uses the following transformation matrices:

$$\Phi_1(A, \delta) = \sum_{i=0}^{\infty} \frac{\delta^{i+1}}{(i+1)!} A^i, \qquad \Phi_2(A, \delta) = \sum_{i=0}^{\infty} \frac{\delta^{i+2}}{(i+2)!} A^i. \tag{3.8}$$

If A is invertible, Φ_1 and Φ_2 can be computed as $\Phi_1(A, \delta) = A^{-1}\left(e^{\delta A} - I\right)$, $\Phi_2(A, \delta) = A^{-2}\left(e^{\delta A} - I - \delta A\right)$. Otherwise, they can be computed as submatrices of the block matrix

$$\begin{pmatrix} e^{A\delta} & \Phi_1(A, \delta) & \Phi_2(A, \delta) \\ 0 & I & I\delta \\ 0 & 0 & I \end{pmatrix} = \exp \begin{pmatrix} A\delta & I\delta & 0 \\ 0 & 0 & I\delta \\ 0 & 0 & 0 \end{pmatrix}.$$

We can now use these operators to obtain a precise over-approximation of $\mathrm{Reach}_{0,\delta}(\mathcal{X}_0)$ and $\mathrm{Reach}_{\delta,\delta}(\{0\})$. We start with a first-order approximation of the latter :

Lemma 3.1. *[44] Let Ψ_δ be the set defined by*

$$\Psi_\delta = \delta\mathcal{U} \oplus \mathcal{E}_\Psi(\delta), \tag{3.9}$$

$$\mathcal{E}_\Psi(\delta) = \boxdot\big(\Phi_2(|A|,\delta) \boxdot (A\mathcal{U})\big). \tag{3.10}$$

Then $\mathcal{Y}_\delta \subseteq \Psi_\delta$.

Our starting point for $\Omega_{0,\delta}$ is a linear interpolation between $\mathcal{X}_0$ and $\mathcal{X}_\delta$ using a parameter $\lambda = t/\delta$ representing normalized time. For each point in time t, $\mathrm{Reach}_{t,t} = \mathrm{Reach}_{\lambda\delta,\lambda\delta}$ is a convex set, for which we construct an over-approximation Ω_λ. Our overapproximation for the time interval $[0,\delta]$ is then the convex hull of all Ω_λ over $\lambda \in [0,1]$. Using a forward, respectively backward, interpolation leads to an error term proportional to λ, respectively $1-\lambda$. Intersecting both approximations gives the following result:

Lemma 3.2. *[44] Let $\lambda \in [0,1]$, and let Ω_λ be the convex set defined by:*

$$\begin{aligned}
\Omega_\lambda &= (1-\lambda)\mathcal{X}_0 \oplus \lambda e^{\delta A}\mathcal{X}_0 \oplus \lambda\delta\mathcal{U} \\
&\quad \oplus \mathcal{E}_\Omega(\delta,\lambda) \oplus \lambda^2\mathcal{E}_\Psi(\delta), \ \textit{with} \\
\mathcal{E}_\Omega(\delta,\lambda) &= \big(\lambda\mathcal{E}_\Omega^+(\delta) \cap (1-\lambda)\mathcal{E}_\Omega^-(\delta)\big) \\
\mathcal{E}_\Omega^+(\delta) &= \boxdot\left(\Phi_2(|A|,\delta) \boxdot \left(A^2\mathcal{X}_0\right)\right), \\
\mathcal{E}_\Omega^-(\delta) &= \boxdot\left(\Phi_2(|A|,\delta) \boxdot \left(A^2e^{\delta A}\mathcal{X}_0\right)\right), \\
\mathcal{E}_\Psi(\delta) &= \boxdot\left(\Phi_2(|A|,\delta) \boxdot (A\mathcal{U})\right).
\end{aligned}$$

Then $\mathcal{X}_{\lambda\delta} \subseteq \Omega_\lambda$ and $\mathcal{X}_{0,\delta} \subseteq \Omega_{0,\delta}$, with

$$\Omega_{0,\delta} = \mathrm{CH}\Big(\bigcup\nolimits_{\lambda\in[0,1]} \Omega_\lambda\Big). \tag{3.11}$$

$\Omega_{0,\delta}$, as defined above, might seem hard to represent. In fact, its support function is not much harder to compute than the one of $\mathcal{X}_0$ and $\mathcal{U}$. Let

$$\begin{aligned}
\omega(\lambda,\ell) = (1-\lambda)\rho_{\mathcal{X}_0}(\ell) + \lambda\rho_{\mathcal{X}_0}\big((e^{\delta A})^\top\ell\big) + \lambda\delta\rho_{\mathcal{U}}(\ell) \\
+ \rho_{\lambda\mathcal{E}_\Omega^+\cap(1-\lambda)\mathcal{E}_\Omega^-}(\ell) + \lambda^2\rho_{\mathcal{E}_\Psi}(\ell).
\end{aligned} \tag{3.12}$$

Since the signs of λ, $(1-\lambda)$, and λ^2 do not change on $[0,1]$, we have:

$$\rho_{\Omega_{0,\delta}}(\ell) = \max_{\lambda\in[0,1]} \omega(\lambda,\ell). \tag{3.13}$$

The support function of $\mathcal{E}_\Omega(\delta,\lambda)$ can easily be expressed as a piecewise linear function of λ. For any λ, this set is a centrally symmetric box and its support function is:

$$\rho_{\mathcal{E}_\Omega(\delta,\lambda)}(\ell) = \sum_{i=1}^{n} \min(\lambda e_i^+, (1-\lambda)e_i^-)|\ell_i|,$$

where vectors e^+ and e^- are such that $\rho_{\mathcal{E}_\Omega^+}(\ell) = |\ell|^\mathsf{T}e^+$ and $\rho_{\mathcal{E}_\Omega^-}(\ell) = |\ell|^\mathsf{T}e^-$. Thus we only have to maximize a piecewise quadratic function in one variable on $[0,1]$ after the evaluation of the support function of the sets involved.

3.2.2 Time Step Adaptation with Error Bounds

Time steps are generally hard to choose, and their value is rarely chosen in itself, but according to an expected precision. In order to efficiently choose our variable time step, we must be able to evaluate the error introduced by time discretization. In our error calculation, we do not bound the error introduced at each step, but the overall error introduced since the beginning of the current continuous evolution. We must take into account the accumulation of errors, not done carefully we might exhaust our error tolerance before the end of the computation and become unable to advance in time without exceeding it.

Our choice of error bound $\varepsilon(\ell)$ is the difference between the computed support function and the support function of the reachable set at time t. For each set Ω_k we define

$$\varepsilon_{\Omega_k}(\ell) = \rho_{\Omega_k}(\ell) - \rho_{\mathcal{X}_{t_k,t_{k+1}}}(\ell) \tag{3.14}$$

The total approximation error is then

$$\varepsilon(\ell) = \max_{k\in 0,\ldots,N-1} \varepsilon_{\Omega_k}(\ell).$$

Note that $\mathcal{X}_{t_k,t_{k+1}}$ is generally not a convex set, and by overapproximating it with the convex set Ω_k we incur an additional error that is not captured by $\varepsilon_{\Omega_k}(\ell)$. The bound $\varepsilon(\ell)$ allows us to decide whether or not the reachable set satisfies a linear constraint $\ell \cdot x \leq b$ (e.g., whether the states surpass a certain threshold) with an uncertainty margin of $\varepsilon(\ell)$.

To compute $\varepsilon_{\Omega_k}(\ell)$, we must take into account the error for Ψ_δ defined as in Lemma 3.1 and $\Omega_{0,\delta}$ defined as in Lemma 3.2. Let

$$\varepsilon_{\Psi_\delta}(\ell) = \rho_{\Psi_\delta}(\ell) - \rho_{\mathcal{Y}_\delta}(\ell), \tag{3.15}$$

$$\varepsilon_{\Omega_{0,\delta}}(\ell) = \rho_{\Omega_{0,\delta}}(\ell) - \rho_{\mathcal{X}_{0,\delta}}(\ell). \tag{3.16}$$

Lemma 3.3. *[44] For any ℓ in $\mathbb{R}^n$:*

$$\varepsilon_{\Psi_\delta}(\ell) \leq \rho_{\mathcal{E}_\Psi(\delta)}(\ell) + \rho_{-A\Phi_2(A,\delta)\mathcal{U}}(\ell) \tag{3.17}$$

$$\begin{aligned}\varepsilon_{\Omega_{0,\delta}}(\ell) \leq \max_{\lambda\in[0,1]} \Big(&\rho_{\lambda\mathcal{E}^+_\Omega(\mathcal{X}_0,\delta)\cap(1-\lambda)\mathcal{E}^-_\Omega(\mathcal{X}_0,\delta)}(\ell) \\ &+\lambda^2\rho_{\mathcal{E}_\Psi(\delta)}(\ell) + \lambda\rho_{-A\Phi_2(A,\delta)\mathcal{U}}(\ell)\Big).\end{aligned} \tag{3.18}$$

Computing the support function of Ω_k as defined by (3.7) and applying the above lemmas as well as time shifting, we obtain $\varepsilon_{\Omega_k}(\ell)$ as follows.

$$\begin{aligned}\varepsilon_{\Psi_{k+1}}(\ell) &= \varepsilon_{\Psi_k}(\ell) + \varepsilon_{\Psi_{\delta_k}}(e^{At_k\top}\ell), \\ \varepsilon_{\Omega_k}(\ell) &= \varepsilon_{\Omega_{0,\delta_k}}(e^{At_k\top}\ell) + \varepsilon_{\Psi_k}(\ell).\end{aligned} \tag{3.19}$$

Given the above error bounds, one can adapt the time steps during the computation of the sequence such that $\varepsilon(\ell)$ is kept arbitrarily small. The problem lies in the error $\varepsilon_{\Psi_k}(\ell)$, which accumulates with k, so that an a-posteriori refinement would require the whole squence to be recomputed. We therefore use the following simple heuristic to compute the sequence of $\rho_{\Omega_k}(\ell)$ for a given template direction ℓ such that $\varepsilon(\ell) \leq \hat{\varepsilon}$ for a given error bound $\hat{\varepsilon} \in \mathbb{R}^{>0}$. Instead of computing $\rho_{\Omega_k}(\ell)$ directly, we first compute the whole sequence $\rho_{\Psi_k}(\ell)$,

then the sequence $\rho_{\Omega_{0,\delta_k}}(e^{At_k}{}^\top \ell)$, and only then combine them to get the sequence $\rho_{\Omega_k}(\ell)$. This separation allows us to choose a separate time step for each sequence, adapting the error as necessary. Additionally, by computing the sequences one after another, the last one can pick up the slack in the error bound of the first sequence.

In the following we suppose that $\hat{\varepsilon}$ is distributed a-priori on both sequences, so that we have $\hat{\varepsilon}_\Omega$ and $\hat{\varepsilon}_\Psi$ in $\mathbb{R}^{>0}$ with $\hat{\varepsilon} = \hat{\varepsilon}_\Omega + \hat{\varepsilon}_\Psi$. This distribution can be established, e.g., by a prior coarse-grained run with a large error bound, or by a run with large fixed time steps. Because the error of Ψ_k accumulates with k, it is chosen (somewhat arbitrarily) to remain below a linearly increasing bound. The error of $e^{At_k}\Omega_{0,\delta_k}$ does not depend on previous computation steps, so it can be adapted on the fly to meet the required bound. We proceed as follows:

1. Compute $\rho_{\Psi_k}(\ell)$ such that $\varepsilon_{\Psi_k}(\ell) \leq \frac{t_k^\Psi}{T}\hat{\varepsilon}_\Psi$. At each step k, we must find a δ_k^Ψ, ideally the biggest, such that:

$$\varepsilon_{\Psi_k}(\ell) + \varepsilon_{\Psi_{\delta_k^\Psi}}(e^{At_k^\Psi}{}^\top \ell) \leq \frac{t_k^\Psi + \delta_k^\Psi}{T}\hat{\varepsilon}_\Psi$$

   Finding δ_k^Ψ is possible because $\varepsilon_{\Psi_\delta}(e^{At}{}^\top \ell) = O(\delta^2)$. First, we fix an initial time step δ_{-1}^Ψ. Then, at each step k, we start a dichotomic search from δ_{k-1}^Ψ along the δ for which sets and matrices involved in the computation of Ψ_δ have already been evaluated, trying new values only when necessary.

2. Compute $\rho_{\Omega_{0,\delta_k^\Omega}}(e^{At_k^\Omega}{}^\top \ell)$ such that

$$\varepsilon_{\Omega_{0,\delta_k^\Omega}}(e^{At_k^\Omega}{}^\top \ell) + \varepsilon_{\Psi_{i(k)}}(\ell) = \varepsilon_{\Omega_k}(\ell) \leq \hat{\varepsilon}$$

   where $i(k)$ is such that $t_{i(k)-1}^\Psi < t_k^\Omega \leq t_{i(k)}^\Psi$. This can be done with a dichotomic search over the sequence of t_k^Ψ already computed for $\rho_{\Psi_k}(\ell)$. If there is a k such that $\delta_{i(k)}^\Psi$ must be further refined, then for the newly introduced indices k_j, we have $i(k_j) = i(k)$.

To combine the above two sequences into $\rho_{\Omega_k}(\ell)$, we use the sequence of time steps δ_k^Ω. If this introduces new times t_k^Ω in the sequence of t_k^Ψ, we can compute the missing values for $\rho_{\Psi_k}(\ell)$ by starting from $t_{i(k)-1}^\Psi$. This does not trigger the recomputation of $\rho_{\Psi_k}(\ell)$ for other time points since the sequence $\varepsilon_{\Psi_k}(\ell)$ is increasing.

3.3 Computing Transition Successors

Each flowpipe that is created by the time elapse step is passed to the computation of transition successors. States that take the transition must satisfy the guard, are then mapped according to the assignment and the result must satisfy the invariant of the target location. Consider a transition e with guard $\mathcal{G}$ and an assignment of the form

$$x' = Rx + w, \qquad w \in \mathcal{W}.$$

Let the invariant of the source location be $\mathcal{I}^-$ and of the target location be $\mathcal{I}^+$. Then the discrete successors of the transition are given by

$$\mathsf{post}_e(\mathcal{X}) = \big(R\big(\mathcal{X} \cap \mathcal{G} \cap \mathcal{I}^-\big) \oplus \mathcal{W}\big) \cap \mathcal{I}^+. \tag{3.20}$$

We now discuss how the operations for this image computation, intersection and assignment, can be carried out efficiently. We assume $\mathcal{G}$,$\mathcal{I}^-$,$\mathcal{I}^+$ to be polyhedra and assume that the set of template directions L contains the normal vectors of the constraints of these polyhedra. To make the intersection of the support function object $\mathcal{X}$ and $\mathcal{G}$,$\mathcal{I}^-$,$\mathcal{I}^+$ scalable, one can compute the outer approximation before the intersection operation, as in [44]. For lack of a better term, we call this the *standard* discrete image operator for template approximation:

$$\mathsf{post}_e^{\mathrm{std}}(\mathcal{X}) = \big\lceil R\big(\lceil \mathcal{X} \rceil_L \cap \mathcal{G} \cap \mathcal{I}^-\big) \oplus \mathcal{W}\big\rceil_L \cap \mathcal{I}^+. \tag{3.21}$$

Since all operators that make up $\mathsf{post}_e\,(\mathcal{X})$ are monotone,

$$\mathsf{post}_e(\mathcal{X}) \subseteq \mathsf{post}_e^{\mathrm{std}}(\mathcal{X}). \tag{3.22}$$

Note that if R is invertible and $\mathcal{W}$ is deterministic (a point), the outermost outer approximation is not necessary since the resulting polyhedron can be computed efficiently with exact methods. With the intersection operator proposed in this paper, we aim at increasing the precision by computing instead of $\mathsf{post}_e^{\mathrm{std}}(\mathcal{X})$

$$\big\lceil R\big(\mathcal{X} \cap \mathcal{G} \cap \mathcal{I}^-\big) \oplus \mathcal{W}\big\rceil_L \cap \mathcal{I}^+. \tag{3.23}$$

To further improve the accuracy of this approximation, we include the pre-image of the target invariant as follows. This can lead to substantial improvements, as shown in Fig. 3.3. Let the target invariant be

$$\mathcal{I}^+ = \Big\{x \;\Big|\; \bigwedge_{i=1}^{m} \bar{a}_i^{\mathsf{T}} x \le \bar{b}_i\Big\}.$$

An overapproximation of the pre-image of $\mathcal{I}^+$ with respect to the assignment $x' = Rx + w$ is given by

$$\mathcal{I}^* = \Big\{x \;\Big|\; \bigwedge_{i=1}^{m} \bar{a}_i^{\mathsf{T}} R x \le \bar{b}_i + \rho_{\mathcal{W}}(-\bar{a}_i)\Big\}. \tag{3.24}$$

Lemma 3.4. $(R\mathcal{X} \oplus \mathcal{W}) \cap \mathcal{I}^+ \subseteq R(\mathcal{X} \cap \mathcal{I}^*) \oplus \mathcal{W}$. *Equality holds if* $\mathcal{W} = \{w\}$.

We obtain our image operator

$$\widehat{\mathsf{post}}_e\,(\mathcal{X}) = \big\lceil R\big(\mathcal{X} \cap \mathcal{G} \cap \mathcal{I}^- \cap \mathcal{I}^*\big) \oplus \mathcal{W}\big\rceil_L \cap \mathcal{I}^+. \tag{3.25}$$

It is straightforward to show that this is a tight overapproximation in the following sense:

Lemma 3.5. $\mathsf{post}_e\,(\mathcal{X}) \subseteq \widehat{\mathsf{post}}_e\,(\mathcal{X})$. *If* $\mathcal{W} = \{w\}$, *then*

$$\widehat{\mathsf{post}}_e\,(\mathcal{X}) = \lceil \mathsf{post}_e\,(\mathcal{X}) \rceil_L \cap \mathcal{I}^+.$$

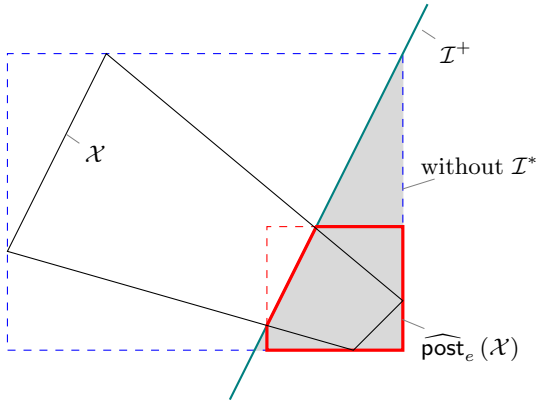


Figure 3.3: The image of $\mathcal{X}$ using the approximation operator (3.25), with the axis directions as template directions. Here, $R = I, \mathcal{W} = 0$, so $\mathcal{I}^+ = \mathcal{I}^*$. Due to the intersection with the pre-image of the target invariant, $\mathcal{I}^*$, the result of (3.25) (shown in thick red) is considerably more accurate than the same approximation without $\mathcal{I}^*$ (shown shaded gray)

Note that $\mathcal{G}, I^-, \mathcal{I}^*$ frequently contain redundant constraints and have matching inequalities that can be simplified to equality constraints. Let $\mathcal{G}^* = \mathcal{G} \cap \mathcal{I}^- \cap \mathcal{I}^*$ be simplified this way. The result of the operator (3.25) is a polyhedral outer approximation. Recalling its definition from (3.4), it involves computing for each $\ell \in L$ the value of the support function,

$$\rho_{R(\mathcal{X}\cap\mathcal{G}^*)\oplus\mathcal{W}}(\ell) = \rho_{\mathcal{X}\cap\mathcal{G}^*}(R^{\mathsf{T}}\ell) + \rho_{\mathcal{W}}(\ell), \tag{3.26}$$

which we obtain exactly or approximately through minimization, as will be discussed in Sect 4.1.

3.4 Clustering

Each flowpipe consists of a possibly large number of convex sets that cover the actual trajectories. When we compute the successor states for a transition, each of these convex sets spawns its own flowpipe in the next time elapse computation. This may multiply the number of sets with each iteration, leading to an explosion in the number of sets and slowing the analysis down to a stall. To avoid this effect and speed up the analysis, we apply what we call *clustering*. Given a hull operator, clustering reduces the number of sets by replacing groups of these sets with a single convex set, their hull. We use the following clustering algorithm for a given hull operator *HULL*. Let the *width* of $\mathcal{P}_1, \ldots, \mathcal{P}_z$ with respect to a direction $\ell \in D$ be

$$\delta_{\mathcal{P}_1,\ldots,\mathcal{P}_z}(\ell) = \max_{i=1..z} \rho(\ell, \mathcal{P}_i) - \min_{i=1..z} \rho(\ell, \mathcal{P}_i). \tag{3.27}$$

Given $\mathcal{P}_1, \ldots, \mathcal{P}_z$ and a *clustering factor* of $0 \leq c \leq 1$, the clustering algorithm produces a set of polyhedra $Q_1, \ldots, Q_r$, $r \leq z$, as follows:

1. Let $i = 1$, $r = 1$, $Q_r := \mathcal{P}_i$.
2. While $i \leq z$ and $\forall \ell \in D : \delta_{Q_r,\mathcal{P}_i}(\ell) \leq c\delta_{\mathcal{P}_1,\ldots,\mathcal{P}_z}(\ell)$, $Q_r := HULL(Q_r, \mathcal{P}_i)$, $i := i + 1$.
3. If $i \leq z$, let $r := r + 1$, $Q_r := \mathcal{P}_i$. Otherwise, stop.

We consider two hull operators: template hull, which is fast but very overapproximative, and convex hull, which is precise but slower. It can be advatageous to combine both, as illustrated by the following example:

Example 3.6. *Consider the 8-variable filtered oscillator from Sect. 3.5.1. At the first discrete state change alone, 57 convex sets can take the transition. Without clustering, the computation is not feasible, as these sets would spawn 57 new flowpipes, and similarly for their successors. Template hull clustering with* $c_{TH} = 0.3$ *produces three sets and results in a total runtime of 11.5s. With* $c_{TH} = 1$ *it produces a single set and takes 3.36s, but with a large overapproximation. Convex hull clustering by itself with* $c_{CH} = 1$ *is very precise but takes 8.19s. Combining both with* $c_{TH} = 0.3, c_{CH} = 1$ *takes 3.64s without noticeable loss in accuracy.*

3.5 Experimental Results

To demonstrate the scalability of our algorithm and the performance of the tool SpaceEx, we present the following experiments. The first system is a simple parameterized system which we use to empirically measure the complexity of our algorithm. The second system is a multivariable continuous control system with complex, tightly coupled dynamics. It illustrates the faithfulness and accuracy of the continuous part of the algorithm. The SpaceEx model files are available at `http://spaceex.imag.fr`.

3.5.1 Filtered Oscillator Model

To measure the performance of our approach, we use a simple parameterized switched oscillator system. The complexity of the system is increased by adding a series of first-order filters to the output x of the oscillator. The filters smooth x, producing a signal z whose amplitude diminishes as the number of filters increases. Note that the dynamics are rather simple, as the filter variables are only weakly coupled with one another. The oscillator is an affine system with variables x, y that switches between two equilibria in order to maintain a stable oscillation, which together with k filters yields a parameterized system with $k + 2$ continuous variables and two locations. One location has the invariant $x \geq -1.4y$, the other $x \leq -1.4y$, and the guards consist of the boundaries of the invariants.

To empirically measure how the complexity depends on the n variables of the system and the m template directions, we run experiments varying just n, just m, and both. *Fixed n:* The average time for a successor computation (discrete followed by continuous) for the 6-variable system over m uniform directions, $m = 8$—256, shows an root mean sqare (RMS) tendency of $O(m^{1.7})$. *Fixed m:* The average time for a successor computation with 200 uniform directions

Table 3.1: Performance results for the filtered oscillator benchmark, varying the number of variables in the system. The time step is $\delta = 0.05$, applying template hull clustering with $c_{TH} = 0.3$ followed by convex hull clustering with $c_{CH} = 1$. Indicated are the runtime, memory and iterations required to compute a fixed-point, and the largest error in any direction in any time step

Variables	Time [s]	Mem. [MB]	Iter.	Error
Box constraints				
18	2,0	9,3	9	0,010
34	9,1	20,2	13	0,010
66	77,3	50,3	23	0,013
130	1185,6	194,3	39	0,030
198	7822,5	511,0	57	0,074
Octagonal constraints				
2	0,7	11,8	6	0.009
4	1,4	11,8	6	0.025
6	4,7	13,3	6	0.025
10	33,0	23,0	7	0.025
18	538,4	67,9	10	0.025

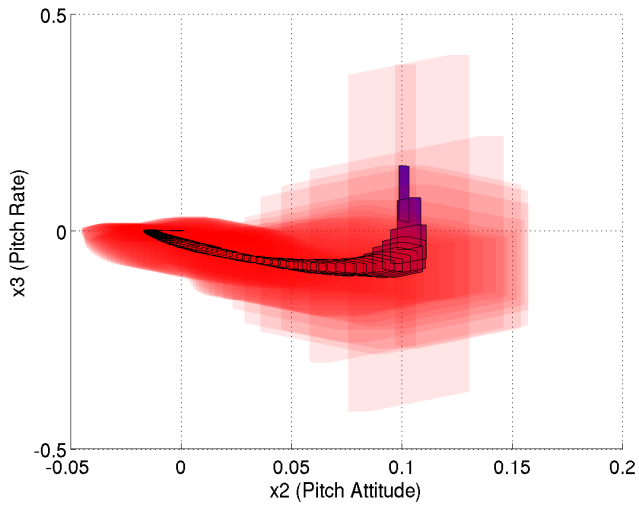
with $n = 6$–16 shows an RMS tendency of $O(n)$. Table 3.1 shows the complete runtime of a fixed-point computation for box and uniform directions for the system with up to 198 variables. The RMS tendency is $O(n^{2.7})$ for box directions and $O(n^{4.7})$ for octagonal directions, which confirms the results for fixed n and fixed m.

3.5.2 Helicopter Controller

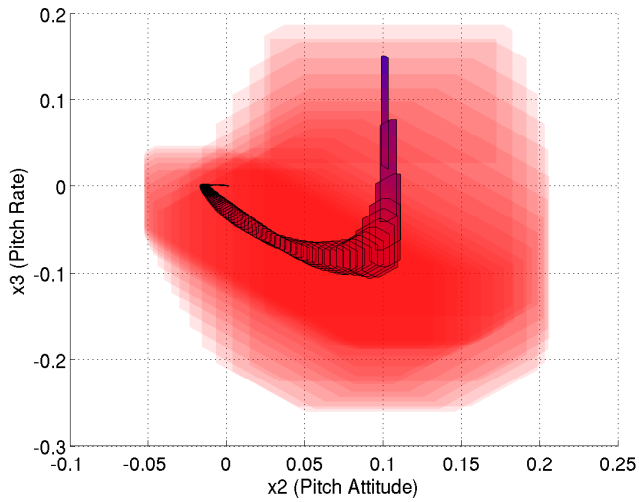
To measure the performance of our algorithm for complex dynamics, we analyze the helicopter controller from [82]. We analyze the controlled plant, a 28-dimensional continuous linear time-invariant (LTI) system. The plant is a small disturbance model of a helicopter, given as an 8-dimensional LTI system. The controller we examine is an $\mathcal{H}_\infty$ mixed-sensitivity design for rejecting atmospheric turbulence, given as a 20-dimensional LTI system.

Figure 3.4 shows the increased accuracy of our new approximation model with respect to previous models. Tables 3.2 and 3.3 show the performance results with fixed time steps and with variable time steps, each for the different error models.

In [82], two different controllers are designed for the helicopter, one of which is specifically tuned for disturbance rejection. Letting the rotor collective be a nondeterministic input in the interval $[-1, 1]$, we compute the reachable states in 5s for one controller and in 14s for the other, as shown in Fig. 3.5.

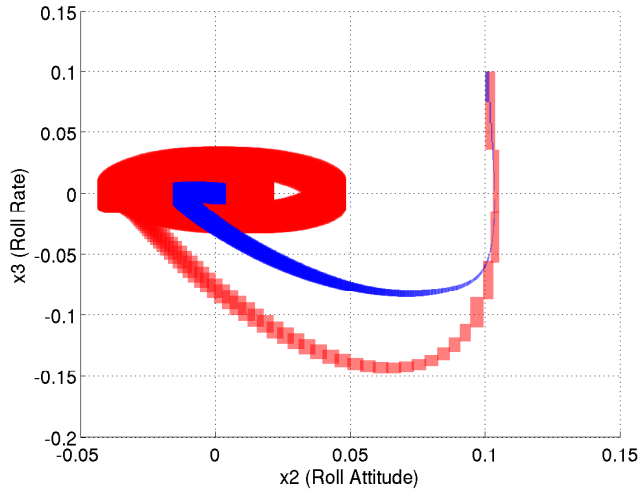


(a) $\delta t = 0.05$ for both

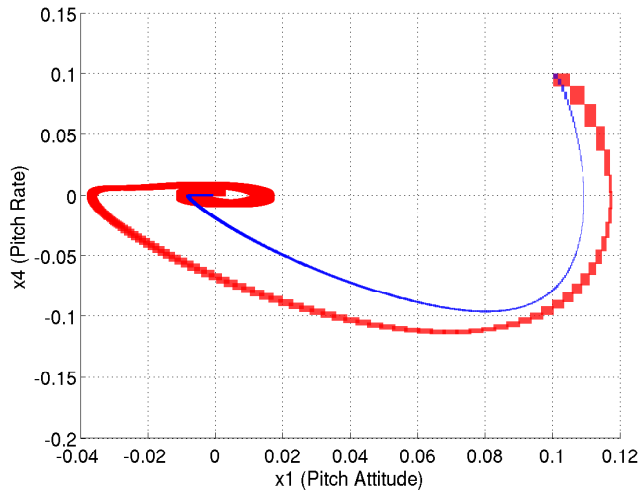


(b) forw: $\delta t = 0.005$, interpfb: $\delta t = 0.05$

Figure 3.4: The reachable states of the 28-variable controlled helicopter system in the plane (x_2, x_3) (corresponding to roll attitude and roll rate), computed with octagonal constraints. We compare the new error scheme (interfb), shown in dark blue, with that of [65] (a) and of [67] (b), shown in bright red.



(a) Roll stabilization



(b) Pitch stabilization

Figure 3.5: Comparison of two disturbance rejection models. Reachable sets with nondeterministic inputs for the helicopter example for the two disturbance rejection models compared in [82] ($T = 20$, method interfb and error tolerance of 0.1 for both). This confirms that the better disturbance rejection model proposed (in blue) actually stabilizes the system faster. However, in [82], this analysis was based only on several simulations.

Table 3.2: Comparison of error models with fixed time-step on the helicopter controller. The error model introduced in this paper (interpfb) clearly outperforms that proposed in [67] (forward) and the one proposed in [65] (interp). The runtime is given in seconds and memory is indicated in MB.

	forward			interp			interpfb		
δt	Mem.	Time	Error	Mem.	Time	Error	Mem.	Time	Error
0.05	9.44	1.48	9.67e+22	9.61	1.60	16.1	9.59	1.65	2.95
0.01	10.5	7.09	3.85e+5	10.5	7.60	0.191	10.5	8.16	0.178
5e-3	10.3	14.1	2.47e+3	10.2	15.2	4.37e-2	12.6	15.8	2.82e-2
1e-3	23.1	71.1	12.4	18.4	76.7	1.59e-3	18.5	78.7	1.07e-3
5e-4	27.9	142	2.56	27.9	155	3.89e-4	28.2	157	2.66e-4

Table 3.3: Comparison of variable time-step vs fixed time-step on the helicopter controller. The variable step implementation outperforms a fixed step scheme even in the *ideal case*, i.e., with the best error model and assuming we know in advance the optimal time step δt to satisfy the error bound. This is always true for the number of steps taken and the slightly higher computational time for some case is explained by frequent changes in choice of the time step.

	Ideal fixed step (interpfb)			var step (interp)		var step (interpfb)	
Err. req.	nb steps	δt used	Time [s]	nb steps	Time [s]	nb steps	time
1	1500	0.02	11.68	1475	12.0	974	8.359
0.1	3418	8.78e-3	26.6	4334	33.9	2943	31.2
0.01	11539	2.6e-3	90.3	14070	108	9785	77.9
1e-3	44978	6.67e-4	351	39152	301	27855	234
1e-4	101352	2.96e-4	902	85953	688	64315	811

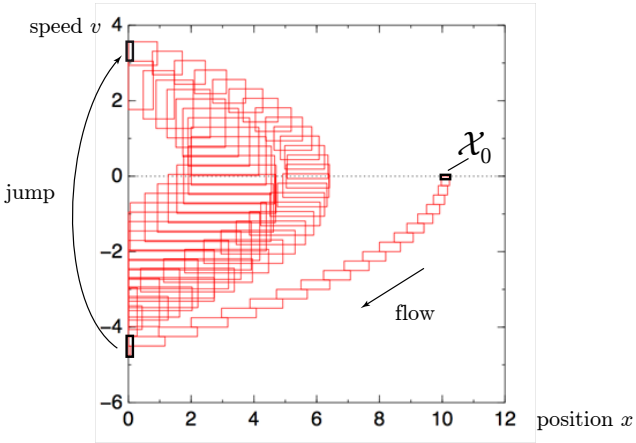
Chapter 4

One-Step Template Refinement

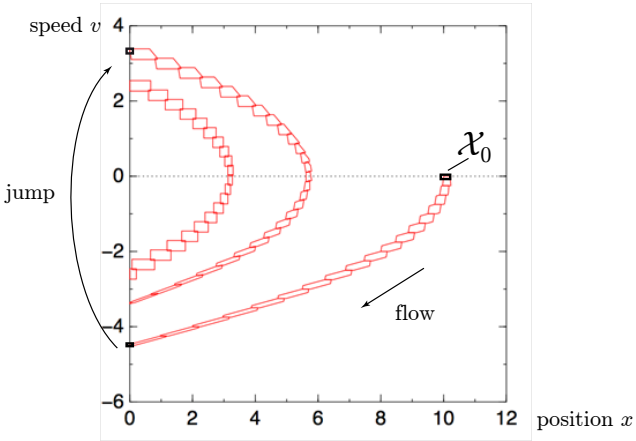
In chapter 3 we presented a scalable technique for computing the reachable states for hybrid systems with piecewise affine dynamics. Simply put, its efficiency hinges on computing an overapproximation in the form of template polyhedra, whose facet normals consist of user-defined template directions. One of the strong points of the algorithm is the precision of time elapse operator. At each application of the operator, the approximation error can be fixed to an arbitrarily small value and does not accumulate with time if the ODEs are deterministic. While restricting the approximation to few template directions, e.g., bounding box directions, makes this approach scalable, it can result in a large approximation error when computing the image of a discrete transition as illustrated in Fig. 4.1(a). The accuracy of the flowpipe-guard intersection has therefore considerable impact on the quality of the computed reachable set.

An approach to more accurately compute the flowpipe-guard intersection for hyperplanar guards was proposed by [66], see also [65, p.114–122]. Their algorithm computes the intersection of a convex set, represented by its support function, with a single hyperplane. This reduces the intersection to the minimization of a unimodal function, for which they propose a dichotomic search and a golden section search algorithm, for more details see Sect. 4.1.

In this chapter, we revisit this approach, generalizing it from hyperplanes to halfspaces and polyhedra. Using a different formulation of the problem, we reduce the intersection to minimizing a *convex* function, which allows us to compute the optimality gap and thus obtain a result of guaranteed accuracy. In the setting of our reachability algorithm, the minimization function is the support function of a polyhedral set and therefore piecewise linear. For this class, our minimization algorithm terminates in finite number of steps for any desired error bound including zero. Similar to [65], we use branch-and-bound techniques and solve these minimization problems simultaneously to avoid redundant computations. In the case of polyhedra, the exact solution leads to a multi-dimensional optimization problem. As an alternative with lower complexity, we propose a per-constraint intersection to which branch-and-bound techniques are readily applied.



(a) Using outer approximations for the intersection with the guard set ($x = 0$), the approximation error increases with each jump, to the point where the computed set diverges when more jumps are added



(b) Our accurate intersection algorithm adds template directions when required by subsequent jump computations. Here, the flowpipe after the second jump has no additional template directions since the third jump has not yet been computed

Figure 4.1: Reachable state computation for two jumps (discrete transitions) of a bouncing ball, without and with accurate intersection, all other parameters being equal. The user-defined template directions are the axis directions, resulting in a bounding-box overapproximation

Our minimization algorithm is similar to sandwich algorithms used in literature mainly for approximation, see [21] and references therein. While the literature on minimizing piecewise linear functions is mostly concerned with convergence properties, our focus is on the result for a very small number of evaluations, since in our application each function evaluation is rather costly.

In the next section we discuss the problem of intersecting a hyperplane, halfspace or polyhedron with a convex set represented by its support function independently of the reachability context. We present our minimization algorithm and compare its performance for hyperplane intersection to the golden section search proposed in [66]. In Sect. 4.2, we apply this technique to the flowpipe-guard intersection problem. Since our flowpipe approximation consists generally of a large number of convex sets, we discuss efficiency improvements such as branch-and-bound techniques. The performance of our algorithm is illustrated by experimental results in Sect. 4.4.

4.1 Intersection of Support Functions with Polyhedra

We consider convex sets represented by their support function, as introduced in Sect. 3.1. While support functions are very efficient for geometric operations like affine maps or convex hull, the intersection operation is difficult, and generally involves solving a high-dimensional optimization problem. In the following, we will present scalable techniques for computing the intersection of such a set with a polyhedron.

Intersection with a Halfspace or Hyperplane We now consider the intersection of a closed and bounded convex set $\mathcal{X}$ with a halfspace or a hyperplane. Later we extend these results to polyhedra. As noted in [65], the support function of the intersection of compact convex sets $\mathcal{X}$ and $\mathcal{Y}$ can be reduced to the minimization problem

$$\rho_{\mathcal{X}\cap\mathcal{Y}}(\ell) = \inf_{v\in\mathbb{R}^n} (\rho_{\mathcal{X}}(\ell - v) + \rho_{\mathcal{Y}}(v)). \tag{4.1}$$

If $\mathcal{Y}$ is a halfspace or a hyperplane, we show that (4.1) simplifies to a univariate minimization problem as follows.

Lemma 4.1. *Consider the halfspace $\mathcal{H} = \{x \mid a^\mathsf{T} x \leq b\}$ and the hyperplane $\mathcal{H}' = \{x \mid a^\mathsf{T} x = b\}$, and let*

$$f(\lambda) = \rho_{\mathcal{X}}(\ell - \lambda a) + \lambda b. \tag{4.2}$$

Then we have

$$\rho_{\mathcal{X}\cap\mathcal{H}}(\ell) = \inf_{\lambda\in\mathbb{R}^{\geq 0}} f(\lambda), \quad \rho_{\mathcal{X}\cap\mathcal{H}'}(\ell) = \inf_{\lambda\in\mathbb{R}} f(\lambda). \tag{4.3}$$

Note that $f(\lambda)$ is convex, since every support function is convex and the sum of two convex functions is convex. If $\mathcal{X}$ is a polyhedron, (4.3) is a parametric linear program (LP), with λ as parameter, and $f(\lambda)$ is continuous, convex, piecewise linear function, see [32].

Lemma 4.2. *The following relationships about the intersection of a closed, bounded and convex set $\mathcal{X}$ with a halfspace $\mathcal{H}$ are relevant:*

1. $\mathcal{X} \cap \mathcal{H} = \emptyset$ *iff* $-\rho_{\mathcal{X}}(-a) > b$.
2. *If* $\rho_{\mathcal{X}}(a) \leq b$, *then* $\rho_{\mathcal{X}\cap\mathcal{H}}(\ell) = \rho_{\mathcal{X}}(\ell)$.
3. $f(\lambda) \to -\infty$ *as* $\lambda \to \infty$ *iff* $\mathcal{X} \cap \mathcal{H} = \emptyset$.
4. *If* $\mathcal{X} \cap \mathcal{H} \neq \emptyset$, *then* $f(\lambda) \geq -\rho_{\mathcal{X}}(-\ell)$.

Sandwich Algorithm We have the following sandwich algorithm to find a sequence of λ_i that converges towards the minimum of $f(\lambda)$, see Fig. 4.2 for an illustration. For each λ_i, we compute the corresponding value $f(\lambda_i)$, and we call $(\lambda_i, f(\lambda_i))$ a *sample* of $f(\lambda)$. We compute a lower bound function $f^-(\lambda) \leq f(\lambda)$, which we update with each newly computed sample. Given two samples $(\lambda_i, f(\lambda_i))$ and $(\lambda_j, f(\lambda_j))$, $\lambda_i < \lambda_j$, the convexity of $f(\lambda)$ implies that the straight line through them,

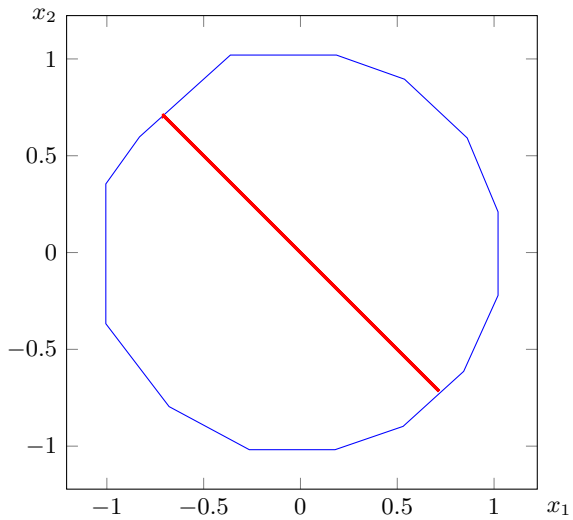
$$f_{ij}^-(\lambda) = \frac{f(\lambda_j) - f(\lambda_i)}{\lambda_j - \lambda_i}(\lambda - \lambda_i) + f(\lambda_i), \tag{4.4}$$

is a lower bound on $f(\lambda)$ to the left and right of the two points, i.e., for all $\lambda \leq \lambda_i$ and $\lambda \geq \lambda_j$, and an upper bound between them, i.e., for $\lambda_i \leq \lambda \leq \lambda_j$. We combine (4.4) for all known samples $(\lambda_i, f(\lambda_i))$ to the following lower bound function, which is defined pointwise over λ:

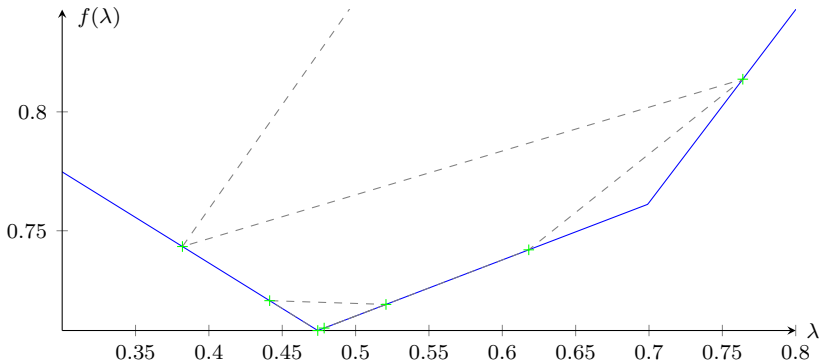
$$f^-(\lambda) = \max\Big(-\infty, \max_{\lambda \leq \lambda_i < \lambda_j} f_{ij}^-(\lambda), \max_{\lambda_i < \lambda_j \leq \lambda} f_{ij}^-(\lambda)\Big). \tag{4.5}$$

We compute an interval $[r_-, r_+]$ containing $\min_{\lambda \in \mathbb{R}^{\geq 0}} f(\lambda)$, whose *optimality gap* $r_+ - r_-$ is smaller than a given threshold $\varepsilon \geq 0$. Our algorithm, called *Lower Bound search*, proceeds as follows:

1. Let $i = 0$, $\lambda_i = 0$, $r_- = -\infty$, $r_+ = +\infty$.
2. Bracket the minimum by adding samples until a turning point is found, i.e., $f(\lambda_{i-1}) \leq f(\lambda_{i-2})$ and $f(\lambda_{i-1}) \leq f(\lambda_i)$, increasing the distance between λ_i exponentially.
3. Compute $f(\lambda_i)$ and tighten the interval bounds
   $r_- \leftarrow \inf_{\lambda \in \mathbb{R}^{\geq 0}} f^-(\lambda)$, $r_+ \leftarrow \min(r_+, f(\lambda_i))$
4. Choose the next sample at the lowest point of $f^-(\lambda)$ unless already visited:
   (a) Let $\lambda_{i+1} \leftarrow \operatorname{arginf}_{\lambda \in \mathbb{R}^{\geq 0}}(f^-(\lambda))$.
   (b) If $\lambda_{i+1} \in \{\lambda_0, \ldots, \lambda_i\}$, let $\lambda_{i+1} \leftarrow (\lambda_{i+1} + \lambda_j)/2$, where λ_j is an appropriate neighboring sample.
5. If $r_+ - r_- > \varepsilon$, let $i \leftarrow i+1$ and go to (3). Otherwise terminate and return the interval $[r_-, r_+]$.



(a) The polytope $\mathcal{P}$ and its intersection with the hyperplane $\mathcal{H}'$



(b) To compute $\rho_{\mathcal{X}\cap\mathcal{H}'}(\ell)$, we minimize $f(\lambda)$. The function and the samples chosen by the Lower Bound search are shown for $\ell = (0, 1)$

Figure 4.2: Intersection of the hyperplane $\mathcal{H}' = \{x + y = 0\}$ with a polytope $\mathcal{P}$ with 15 facets

Table 4.1: Average performance of Lower Bound search vs GSPD (fixed to 14 function evaluations), intersecting a hyperplane with a polytope

	Lower Bound			GSPD		
facets	samples	err	runtime [ms]	samples	err$\times 10^{-4}$	runtime [ms]
4	6.741	0	0.15	14	8.197	0.71
8	8.523	0	0.34	14	3.200	0.82
16	9.611	0	0.50	14	1.612	1.27
24	10.222	0	0.74	14	1.111	1.80

If $f(\lambda)$ is piecewise linear with a finite number of pieces, the above algorithm terminates with a finite number of steps for every $\varepsilon \geq 0$. This is the case if $\mathcal{X}$ is a polytope.

Our implementation contains further steps to optimally reduce the number of samples and to account for floating point errors. We refer the reader for further details to [79].

Related Work: To the best of our knowledge, this is the first published solution of the intersection with a halfspace or polyhedron. It is derived from previous work on the intersection with a hyper*plane* in [66]. There, computing the support function of the intersection is reduced to a univariate minimization problem that is derived geometrically. Its parameter $\theta \in (0, \pi)$ describes the angle between the sample direction and the normal vector of the hyperplane $\mathcal{H}' = \{x \mid a^\mathsf{T} x = b\}$:

$$\rho_{\mathcal{X} \cap \mathcal{H}'}(\ell) = \inf_{\theta \in (0,\pi)} \frac{\rho_{\mathcal{X}}(\ell \sin\theta + a\cos\theta) - b\cos\theta}{\sin\theta}. \tag{4.6}$$

While (4.6) has the advantage over (4.3) that its argument ranges over a finite interval, its cost function is only known to be unimodal instead of convex. We refer to the approximate solution of (4.6) by golden section search as *Golden Section Search in the Polar Domain* (GSPD).

Experiments: The following experiments shall illustrate the performance of Lower Bound search in comparison with GSPD. The results in this paper were obtained on a standard x86 machine with 32bit operating system. To measure the computation error $err = r^+ - \inf_{\lambda \in \mathbb{R}} f(\lambda)$, we compare the different output values to the result of the Lower Bound search for $\varepsilon = 0$.

Table 4.1 compares the support function computation of the intersection between a regular n-polyhedron in two dimensions with the line $x\cos\theta + y\sin\theta = 0$ in the direction $[x = 0, y = 1]$, for 1000 uniformly distributed $\theta \in [0, \pi]$ by GSPD and our Lower Bound search. The table shows the averaged results. Note that the error of GSPD decreases as the number of facets increases. The reason is that the function becomes flatter near the minimum for a larger number of facets. Hence for a fixed interval in the function domain bracketing the minimum, the difference between the minimum and the upper bound decreases.

Table 4.2 compares the intersection between a regular n-polyhedron with the line $x\cos\theta + y\sin\theta = 0$ in the direction $[x = 0, y = 1]$, for 1000 uniformly distributed $\theta \in [0, \pi]$ for a fixed number of samples. The table shows the averaged results.

Table 4.2: Average performance of Lower Bound search vs GSPD given 6 function evaluations, intersecting a hyperplane with a polytope

	Lower Bound			GSPD	
facets	opt. gap	err	runtime [ms]	err	runtime [ms]
4	0.0338614	0.0285933	0.16	0.0351516	0.25
8	0.0274455	0.0107857	0.10	0.0228235	0.32
16	0.0298651	0.0063944	0.28	0.0156476	0.51
24	0.0302147	0.0044049	0.47	0.0131288	0.98

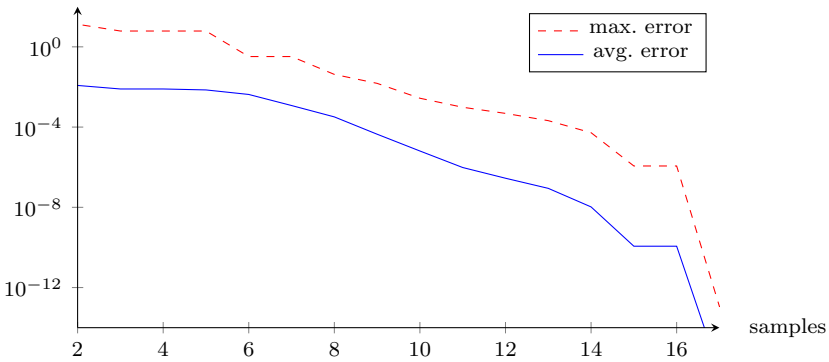


Figure 4.3: Approximation error over the number of samples for the intersection of random halfspaces with random polytopes with 16 facets

Remark 4.3. *Note that in Table 4.2 the computation times differ even though the same number of samples is computed for both LB search and GSPD. Indeed the computation time of a sample is data as well as state dependent. In particular, the LP solver computing the support function keeps its state between calls. A sample can therefore be computed faster if its optimal solution for the corresponding direction is close to the one computed in the previous call.*

Figure 4.3 shows the approximation error of the intersection with a halfspace as a function of the number of samples taken. We measure the absolute error over 10000 random instances of a polytope with 16 facets intersected with a halfspace. The polytope and the intersection are by construction nonempty and the halfspace is nonredundant. After 17 samples, both maximum and average error are below 10^{-13}, which is about as close as we expect given machine precision.

Intersection with a Polyhedron Since a polyhedron is an intersection of halfspaces, we can apply (4.3) repeatedly to obtain the support function of the intersection of a set $\mathcal{X}$ with a polyhedron $\mathcal{P}$:

Lemma 4.4.

$$\rho_{\mathcal{X}\cap\mathcal{P}}(\ell) = \inf_{\lambda\in\mathbb{R}^m,\lambda\geq 0} \rho_{\mathcal{X}}(\ell - \sum_i \lambda_i a_i) + \sum_i \lambda_i b_i. \tag{4.7}$$

This is a convex minimization problem over m variables, where m is the number of constraints in $\mathcal{P}$.

In our implementation, we compute the intersection with each halfspace separately, which allows us to apply the results from the previous section. We intersect $\mathcal{X}$ with each halfspace of $\mathcal{P}$ separately, and combine the results in the approximation

$$\rho_{\mathcal{X}\cap\mathcal{P}}(\ell)^+ = \min_{i=1,\ldots,m} \rho_{\mathcal{X}\cap\{a_i^\mathsf{T} x\leq b_i\}}(\ell). \tag{4.8}$$

Since $\mathcal{X}\cap\mathcal{P}$ is contained in all of the sets $\mathcal{X}\cap\{a_i^\mathsf{T} x \leq b_i\}$,

$$\rho_{\mathcal{X}\cap\mathcal{P}}(\ell) \leq \rho_{\mathcal{X}\cap\{a_i^\mathsf{T} x\leq b_i\}}(\ell).$$

Consequently, $\rho_{\mathcal{X}\cap\mathcal{P}}(\ell) \leq \rho_{\mathcal{X}\cap\mathcal{P}}(\ell)^+$, so we are sure to obtain an overapproximation. An outer approximation computed with $\rho_{\mathcal{X}\cap\mathcal{P}}(\ell)^+$ may be non-empty even though $\mathcal{X}\cap\mathcal{P}$ is empty.

4.2 Flowpipe-Guard Intersection

We now apply the results from the previous section in the reachability computation of a hybrid system. We use the reachability algorithm from Sect. 3.2 to compute a sequence of closed and bounded convex sets $\Omega_0,\ldots,\Omega_N$ that covers the flowpipe starting from $\mathcal{X}_0$, with convex inputs $\mathcal{U}$. We denote this operation by

$$(\Omega_0,\ldots,\Omega_N) = \mathsf{post}_c\left(\mathcal{X}_0,\mathcal{U}\right). \tag{4.9}$$

Each Ω_i is the result of convex hull and Minkowski sum operations on polytopes. So Ω_i is by construction a polytope, but such that computing its constraint representation would be prohibitively expensive. Its support function can, however, be computed efficiently for any given direction. The computation of the sequence Ω_i amounts to a symbolic integration of the ODEs, see (3.5), so the values of Ω_i depend on the values of Ω_{i-1}, etc. This gives us the following limitation, which becomes important when considering intersections:

Assumption 1. *To compute $\rho_{\Omega_i}(\ell)$ for given index i, we also need to compute all preceding values, i.e., $\rho_{\Omega_j}(\ell)$ for $j = 0,\ldots,i-1$.*

There is a partial remedy to this problem. Consider the case where we are interested in computing a subsequence of the flowpipe approximation Ω_i, say for $i\in[c,d]$. This could be, e.g., sets that may take a transition. Under Assumption 1 this requires us to compute the $d+1$ sets with $i\in[0,d]$. We can reduce this computational burden as follows. The sequence Ω_i is constructed such that each set covers the flowpipe over a known time interval $[t_i,t_{i+1}]$. We decompose the system into its autonomous dynamics ($\mathcal{U}=\emptyset$) and its input dynamics ($\mathcal{X}=\emptyset$). Recall that for autonomous dynamics, the set of states reached at exactly time t_c is $\mathcal{X}_{t_c} = e^{At_c}\mathcal{X}$. Starting the flowpipe computation

for the autonomous dynamics from $t = t_c$ instead of $t = 0$, we end up with fewer sets to compute. Let

$$(\Omega_c^x, \ldots, \Omega_d^x) = \mathsf{post}_c\left(e^{At_c}\mathcal{X}, \emptyset\right), \tag{4.10}$$

$$(\Omega_0^u, \ldots, \Omega_c^u, \ldots, \Omega_d^u) = \mathsf{post}_c\left(\emptyset, \mathcal{U}\right), \tag{4.11}$$

such that Ω_i^x and Ω_i^u cover the respective flowpipe on the same time interval $[t_i, t_{i+1}]$. Then using the superposition principle we have that $\Omega_i^x \oplus \Omega_i^u$ covers the flowpipe of $\mathcal{X}_0$ and $\mathcal{U}$ on the time interval $[t_i, t_{i+1}]$. This means we only need to compute the $d - c + 1$ sets of (4.10). While (4.11) still requires the computation of $d+1$ sets, the set $\mathcal{U}$ is in practice often simple, e.g., a hyperbox, so that its support function can be computed much quicker than that of $\mathcal{X}_0$.

Intersection with a Halfspace/Hyperplane In our reachability algorithm, we need to apply the discrete image operator from the previous section to the flowpipe approximation $\Omega_0, \ldots, \Omega_N$, where N is possibly very large. For now we assume that the invariants and the guard are such that $\mathcal{G}^* = \mathcal{G} \cap \mathcal{I}^- \cap \mathcal{I}^*$ is the halfspace

$$\mathcal{G}^* = \{\bar{a}^\mathsf{T} x \leq \bar{b}\}.$$

The extension to hyperplanes is straightforward, as only the domain of the parameter λ changes in (4.3).

According to (3.25), the image is nonempty only for Ω_i where $\Omega_i \cap \mathcal{G}^*$ is nonempty. So with Lemma 4.2, we can limit ourselves to Ω_i with indices in

$$I_{jump} = \{i \mid -\rho_{\Omega_i}(-\bar{a}) \leq \bar{b}\}. \tag{4.12}$$

The result of the discrete image computation is

$$\bigcup_{i \in I} \widehat{\mathsf{post}}_e\left(\Omega_i\right). \tag{4.13}$$

According to (3.26), we need to compute for each $\widehat{\mathsf{post}}_e\left(\Omega_i\right)$ the support $\rho_{\Omega_i \cap \mathcal{G}^*}(\ell)$ for all $\ell \in R^\mathsf{T} L$. Applying the approach from Sect 4.1, this involves minimizing for each Ω_i an instance of (4.2), i.e.,

$$f^i(\lambda) = \rho_{\Omega_i}(\ell - \lambda\bar{a}) + \lambda\bar{b}. \tag{4.14}$$

Recall that according to Assumption 1, computing $\rho_{\Omega_i}(\ell)$ requires computing $\rho_{\Omega_j}(\ell)$ for $j = 0, \ldots, i-1$. Therefore running a minimization algorithm on $f^i(\lambda)$ also produces samples of $f^j(\lambda)$, $j = 0, \ldots, i-1$. These samples can be used to improve the estimate of the minimum of $f^j(\lambda)$. In addition, one can aim at choosing the next λ such that as many of these estimates as possible benefit from the new sample. Our algorithm proceeds as follows to compute the $\min f^i(\lambda)$ up to a given error $\varepsilon \geq 0$:

1. We start with a work list $I_{work} = I_{jump}$, and a first sample at $\lambda_{next} = 0$.
2. Compute $\rho_{\Omega_i}(\ell - \lambda_{next}\bar{a})$ for $i = 0, \ldots, \max(I_{work})$.
3. Update bounds on minima and requests of next λ for each $f^i(\lambda)$:
   Compute $(r_-^i, r_+^i, \lambda_{next}^i)$ for $i \in I_{work}$

4. $\lambda_{next} = select(\{\lambda^i_{next}\}_i)$

5. Keep only problems on the work list whose bounds exceed the error: $I_{work} = \{i \mid r^i_+ - r^i_- > \varepsilon\}$.

6. If $I_{work} \neq \emptyset$, go to (2).

The output of the algorithm are the r^i_+, i.e., an upper bound on the minimum for each $f^i(\lambda)$. The function *select* chooses one of the candidates λ^i_{next} for the next sampling point. We consider picking the first, the last, the median and a random candidate.

Intersection with a Polyhedron The results from the previous section can be used to compute the intersection of a flowpipe with a polyhedron. To trade accuracy against performance, we follow the lines of Sect. 4.1 and intersect with each halfspace of the polyhedron separately. Our goal is to intersect the convex hull of the flowpipe Ω_i in the kth interval I_k with the set

$$\mathcal{G}^* = \left\{ x \;\middle|\; \bigwedge_{j=1}^{m} \bar{a}_j^\mathsf{T} x \leq \bar{b}_j \right\},$$

similar to the intersection with a single halfspace in (4.19). For the intersection of Ω_i the jth halfspace in $\mathcal{G}^*$, we must minimize the function

$$f^i_j(\lambda) = \rho_{\Omega_i}(\ell - \lambda \bar{a}_j) + \lambda \bar{b}_j. \tag{4.15}$$

Applying the same approximation for polyhedron intersection as in (4.8), we obtain the overapproximation

$$\rho_{\mathcal{Y}_k}(\ell) \leq \max_{i \in I_k} \min_{j=1,\ldots,m} \inf_{\lambda \in \mathbb{R}^{\geq 0}} f^i_j(\lambda). \tag{4.16}$$

As with (4.19), we can use a branch-and-bound algorithm to eliminate instances of i, j for which the upper bound of $f^i_j(\lambda)$ is lower than the largest lower bound.

4.3 Branch-and-Bound Clustering

Computing the one-to-one image of the sets covering the flowpipe, as in (4.13), can have the devastating effect of increasing the number of convex sets exponentially with the search depth. To avoid an explosion in the number of sets and gain efficiency, we compute the convex hull of subsets of these sets instead. This is referred to as convex hull *clustering*, see Sect. 3.4 for details. We now discuss how to compute the support function of these clusters efficiently using a branch-and-bound approach.

For any set $\mathcal{X} \subseteq \mathbb{R}^n$, let $\mathrm{CH}(\mathcal{X}) \subseteq \mathbb{R}^n$ denote the *convex hull*

$$\mathrm{CH}(\mathcal{X}) = \left\{ \sum_{i=1}^{n+1} \lambda_i v_i \;\middle|\; v_i \in X, \lambda_i \in \mathbb{R}^{\geq 0}, \sum_{i=1}^{n+1} \lambda_i = 1 \right\}.$$

Let $I_0, \ldots, I_K$ be the maximal connected subsets of I_{jump} i.e., each I_j is the set of indices of a connected subsequence of Ω_i that can take the transition. We compute the outer approximation of the convex hull of these sets,

$$\mathcal{Y}_k = \mathrm{CH}\Big(\bigcup_{i \in I_k} \Omega_i\Big). \tag{4.17}$$

The final result is the outer approximation of the image of the discrete transition,

$$\mathcal{Z}_k = \widehat{\mathsf{post}}_e\left(\mathcal{Y}_k\right). \tag{4.18}$$

With (3.3),(4.3),(3.26) and (4.14), this requires computing for all directions $\ell \in R^{\mathsf{T}}L$

$$\rho_{\mathcal{Y}_k}(\ell) = \max_{i \in I_k} \inf_{\lambda \in \mathbb{R}^{\geq 0}} f^i(\lambda). \tag{4.19}$$

Similarly to above, we accelerate the computation of (4.19) by solving the minimization problems for $i \in I_k$ in parallel, and applying the following improvements:

- updating the estimates for all $f^i(\lambda)$ with every sample, as warranted by Assumption 1,
- using a branch-and-bound algorithm to eliminate the f^i for which the upper bound is lower than the currently largest lower bound.

We provide experimental comparison of the different techniques in the following section.

4.4 Experimental Results

A few benchmarks shall illustrate the performance and precision of the proposed algorithms.

4.4.1 Timed bouncing ball

We take advantage of the simplicity of the timed bouncing ball to compare accuracy and speed of the different intersection variants. The timed bouncing ball has the state variables position x, speed v, and time t. Its hybrid automaton model consists of a single location with invariant $x \geq 0$ and continuous dynamics

$$\dot{x} = v, \qquad \dot{v} = -g, \qquad \dot{t} = 1,$$

where g is the gravitational constant (here normed to 1). A discrete transition from the location to itself changes the sign of th velocity when the ball touches the ground. Here we chose the guard constraints $x \leq 0$ and $v < 0$ (the latter to keep the velocity from flipping when the ball goes upwards), and the assignment

$$x' = x, \qquad v' = -cv, \qquad t' = t,$$

where c is a constant for the loss of speed (here 0.75).

Table 4.3: Speed versus accuracy comparison of different variants of the discrete image computation, applied to the timed bouncing ball example. The accuracy shows in the height of the last jump

direction	ε	clustering	runtime [s]	height x
standard discrete image computation				
box		TH^+	1.3	3.054
box		TH & CH^+	2.6	2.209
box		CH^+	31.2	2.016
oct		TH^+	3.0	0.972
oct		TH & CH^+	12.7	0.901
oct		CH^+	36.6	0.844
discrete image with precise intersection				
box	0.0	TH^+	1.3	1.080
box	0.0	TH & CH^+	3.4	1.017
box	0.0	CH^+	55.9	0.904
precise intersection of convex hull with branch & bound				
box	1.0	CH^-	0.8	1.175
box	0.1	CH^-	0.6	0.815
box	0.0	CH^-	0.6	0.807

In order to avoid an explosion in the number of sets, we cluster the convex sets before (-) or after (+) we compute the image of the discrete transition. We consider as alternatives the template hull of all sets (TH), the convex hull of all sets (CH), and a mix of both (template hull of about 30%, then convex hull). These alternatives have different speed/accuracy trade-offs. Table 4.3 shows the performance results for computing the reachable states over five jumps. As a measure of accuracy, we take the height of the last jump. The time step is fixed at $\delta = 0.01$, i.e., each convex set Ω_i in the flowpipe approximations covers the flowpipe over the time span $[t_i, t_i + \delta]$.

The standard variant applies the outer approximation before the intersection, as in (3.21). Using box directions, the error is so large that the flowpipe of the 5th jump is reaches higher than the 4th, and adding further jumps the computed sets diverge to infinity. Using octagonal directions improves the precision, but slows down the analysis as 18 directions have to be computed instead of 6. Note that the convex hull clustering for octagonal directions is not much slower than for box directions because fewer sets intersect. But even with octagonal directions and very small time steps, the precision leaves to be desired.

The precise variant of the image computation consists of the image operator (3.25) using Lower Bound search with error bound ε. It shows better accuracy than the standard variant, but convex hull clustering comes at a loss in speed. As we do not detect a significant gain in speed from increasing the error bound, only results for $\varepsilon = 0$ are shown.

The precise branch & bound variant shows both the highest accuracy and the greatest speed. It uses the convex approximation of (3.25) indicated in (4.19). Its major gain in performance comes from applying (4.10) and (4.11), which reduces the number of sets in the computation. Increasing the error of the intersection computation reduces somewhat the number of samples, but the time gain is not substantial, see also Remark 4.3.

4.4.2 Filtered Oscillator Model

For a scalability comparison we turn to the filtered oscillator proposed by [44]. It consists of a switched linear oscillator with two state variables x, y and four locations that is attached to K first-order filters put in sequence. The filter stack produces the smoothened output signal z. The total number of state variables is therefore $K + 2$.

Table 4.4 shows results for up to 130 state variables, for both standard discrete image computation and the proposed variant with precise intersection. All instances are computed using box directions. For all except the lower dimensional versions, the precise intersection variant outperforms the standard operator in precision, and often also in speed. In this example, the capacity to compute the intersection up to a given error (column 3) shows its benefits: a small but not too small error greatly reduces the analysis time, at an acceptable loss in accuracy.

Table 4.4: Speed versus accuracy comparison of different variants of the discrete image computation, for computing a fixed-point of the filtered oscillator example. The accuracy shows in the max amplitude of the output signal z

vars	δ	ε	clustering	runtime(s)	max. z	iter
standard discrete image computation						
6	0.01		TH^+	0.3	0.570	5
18	0.01		TH^+	2.1	0.361	9
34	0.01		TH^+	8.7	0.243	13
66	0.05		TH^+	17.4	0.291	23
130	0.05		TH^+	132.7	0.569	39
130	0.025		TH^+	206.0	0.166	41
precise intersection of convex hull with branch & bound						
6	0.01	0	CH^-	0.4	0.567	5
18	0.01	0	CH^-	2.4	0.356	9
34	0.01	0	CH^-	9.0	0.237	14
66	0.05	0.1	CH^-	17.3	0.243	23
66	0.05	0.01	CH^-	18.1	0.232	24
66	0.05	0.001	CH^-	27.4	0.192	37
66	0.05	0	CH^-	55.6	0.190	71
130	0.05	0.1	CH^-	126.0	0.339	39
130	0.05	0.01	CH^-	126.5	0.314	39
130	0.05	0.001	CH^-	205.5	0.190	39
130	0.025	0.01	CH^-	174.2	0.128	65

Chapter 5

Semi-Template Reachability in Space-Time

The reachability algorithms of the previous chapters, as well as related approaches [9, 27, 49, 62, 66], have in common that flowpipes (states reachable over time) are overapproximated with a finite but frequently large number of convex sets. Each of the convex sets covers the flowpipe on a certain time interval, and the approximation error usually increases rapidly with the size of this interval. Often, the time step has to be made very small to achieve a desired accuracy. This in turn may lead to a very large number of convex sets that, depending on the processing or further image computation to be performed, can quickly become prohibitive. For reachability analysis of hybrid systems in particular, we have often observed a fatal explosion in the number of sets when more than a few of the convex sets can take a discrete transition, as each will spawn a new flowpipe in the next state, and so on.

This chapter addresses this fundamental problem from two angles: Firstly, we propose a representation, which we call a flowpipe sampling, that consists of a set of continuous, interval-valued functions over time. A flowpipe sampling attributes to each time point a polyhedral enclosure of the set of states reachable at that time point, thus capable of representing a nonconvex enclosure of a nonconvex flowpipe. This representation helps to decouple, as far as possible, the accuracy from the number of convex sets created in the end. Secondly, we propose a clustering procedure that aims to minimize the number of convex sets produced for a desired accuracy and does so using accuracy bounds established by the flowpipe construction. A-posteriori error measurements help evaluate the distance of the approximation to the actual flowpipe.

The following examples shall illustrate different aspects of the flowpipe approximation problem, as well as showcase the performance of our proposed solution.

Example 5.1 (Helicopter)**.** *Figure 5.1(a) show a flowpipe approximation for an affine helicopter model with 28 state variables plus a clock variable. It was obtained using the approach in Chapter 3, which constructs for each time-step a convex polyhedron in the 29-dimensional state space. The facet normals of the polyhedra, also called* template directions*, are given by the user. In this case, the axis directions are used so that the polyhedra are boxes. The complex dynamics*

of the system require using a very small time step throughout the time horizon of 30 s. Note that only the projection on two of the 29 variables is shown (the vertical speed and the clock), while the approximation takes the variation of all variables into account. As a result of the small time step, 1440 convex sets are constructed for a given directional error estimate *of* $\epsilon = 0.025$. *The construction itself is computationally cheap at 5.9 s CPU time, but the sheer number of sets makes further processing and image computation impractical.*
The approach proposed in this chapter combines an enhanced flowpipe approximation with adaptive clustering that guarantees a conservative error bound on the directional distance to the actual reachable set at each point in time. The time step is adapted separately for each of the template directions and can therefore be considerably larger. In the directions corresponding to the axis of the clock the system evolves linearly, so the time step spans the entire time horizon. The clustering step produces the 32 polyhedra shown in Fig. 5.1(b) for a given directional error bound *of* $\epsilon = 0.025$. *The construction of the flowpipe sampling takes 9.4 s and the clustering and outer polyhedral approximation 4.8 s.*

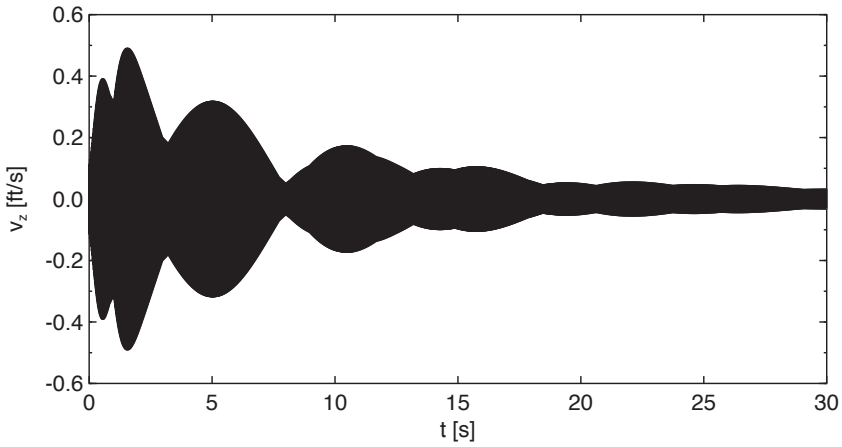
As the following example illustrates, the convexification error is by no means restricted to complex dynamics.

Example 5.2 (Hourglass)**.** *Consider the simple linear ODE system* $\dot{x} = 0$, $\dot{y} = x$, *with initial states* $\mathcal{X}_0 = \{-1 \leq x(0) \leq 1, y(0) = 0\}$ *as shown in Fig. 5.2(a). We consider the states reachable up to time* $t = 1$. *The reachable set is pointwise convex in time, but every convex set that covers the reachable states over a nonsingular time interval is forcibly an overapproximation. The time-step adaptation of the LGG-algorithm in Chapter 3 decreases the time step until all error terms fall below the desired threshold. In this case, the error terms are zero since the trajectories are linear, i.e.,* $x(t) = x(0), y(t) = t \cdot x(0)$. *The LGG-algorithm therefore covers the whole flowpipe in a single timestep. In the best case (arbitrarily well-chosen template directions), the result is the convex hull of the flowpipe, shown in Fig. 5.2(a).*

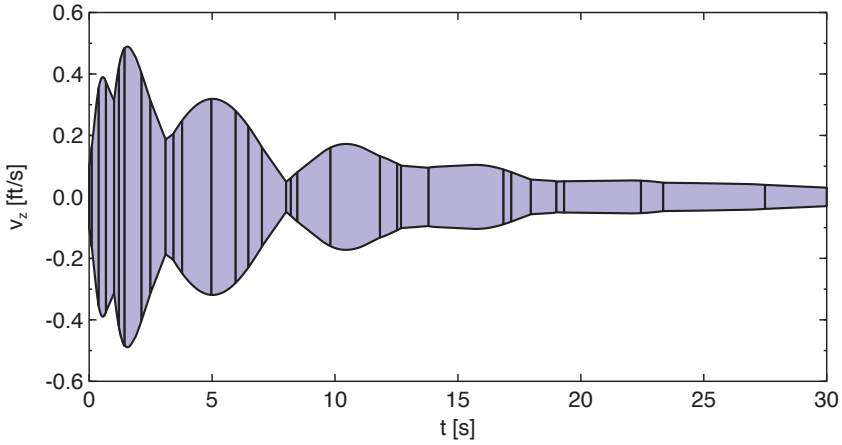
The flowpipe approximation proposed in this chapter can produce a result of arbitrary accuracy in terms of a directional error that is measured in the template directions. As more template directions are added, this directional error converges towards the Hausdorff distance between the actual reachable states and the overapproximation. Figure 5.2(b) shows the result for a given error bound 0.1 *in 64 uniformly distributed template directions.*

The basis of our approach is the representation of a convex set by its support function. Simply put, given the normal vector of a halfspace (direction), the support function tells the position where the halfspace touches the set such that it contains it. If the dynamics are affine and the set of initial states is convex, the states reachable at a given point in time is also convex and can be represented by their support function. The flowpipe can therefore be described by a family of support functions parameterized by time, which has been considered in, e.g., [87, 68]. Our flowpipe sampling builds on this concept, which we refine by accounting for the fact that we can only compute a finite number of values in terms of both direction and time points. There are methods to approximate flowpipes directly with sets that are nonconvex, e.g., with polynomial tubes [78].

The flowpipe construction in this chapter builds heavily on previous work in [66, 44]. Notably, this chapter provides error measurements that include

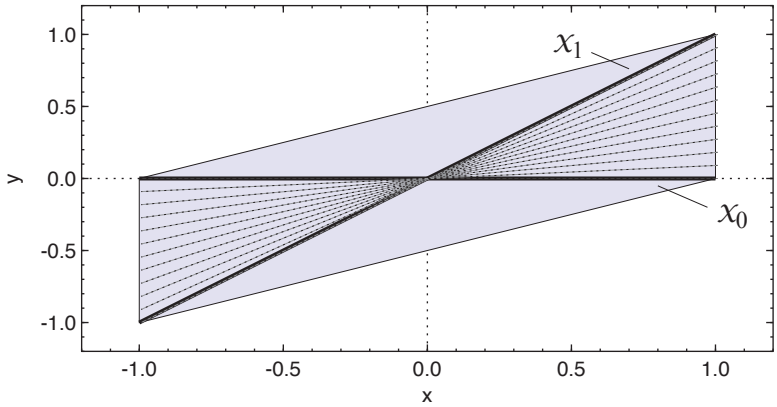


(a) The flowpipe approximation from Chapter 3 constructs one convex polyhedra per time-step, in total 1440 polyhedra

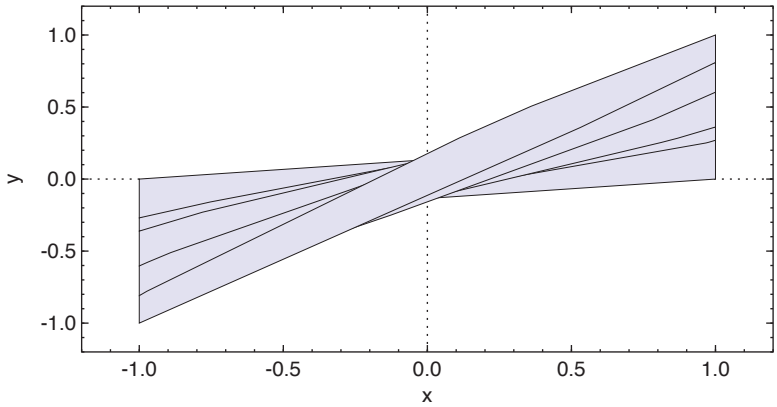


(b) The flowpipe approximation with clustering proposed in this chapter constructs 32 convex polyhedra

Figure 5.1: Flowpipe approximations for the 28-dimensional affine helicopter model from Ex. 5.1, plus a clock. The shown sets are projections of 29-dimensional polyhedra onto two variables, the vertical speed and the clock



(a) The initial set $\mathcal{X}_0$, the final set $\mathcal{X}_1$, and the smallest convex approximation of the reachable set $\mathcal{X}_{[0,1]}$ (shaded)



(b) An approximation of $\mathcal{X}_{[0,t]}$ constructed by the proposed technique for a given error bound 0.1

Figure 5.2: Example 5.2 has a nonconvex reachable set in the shape of an hourglass. It is pointwise convex in time, but every convex set covering a nonsingular interval is forcibly an overapproximation. All trajectories are linear, so the approximation error is impossible to detect based on the dynamics alone

the convexification error. For more detailed comments, see Sect. 5.1.4. The clustering approach of Sect. 5.2 is, to the best of our knowledge, novel.

In the next section, we define flowpipe samplings as a general means to describe and approximate flowpipes and then present our flowpipe approximation algorithm. We also show how a flowpipe sampling can be translated into a set of convex polyhedra. In Sect. 5.2, we present our clustering approach, which minimizes the number convex polyhedra that a flowpipe sampling defines. Section 5.3 presents experimental results.

5.1 Flowpipe Approximation in Space-Time

We consider continuous dynamical systems given by differential equations

$$\dot{x}(t) = Ax(t) + u(t), \qquad u(t) \in \mathcal{U}, \tag{5.1}$$

where $x(t) \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$. The closed, bounded, and convex set $\mathcal{U} \subseteq \mathbb{R}^n$ can represent, e.g., nondeterministic inputs, disturbances or approximation errors. The initial states of the system are given as a convex compact set $\mathcal{X}_0$, i.e., $x(0) \in \mathcal{X}_0$. We refer to the states reachable from $\mathcal{X}_0$ by time elapse as the *flowpipe* of $\mathcal{X}_0$. We compute a sequence of closed and bounded convex sets $\Omega_0, \ldots, \Omega_N$ that covers the flowpipe using an extension of the approximation technique in Chapter 3. In our construction, each Ω_i is the result of convex hull and Minkowski sum operations on polytopes. So Ω_i is itself a polytope, but explicitly computing it would be prohibitively expensive in higher dimensions. Its support function can, however, be computed efficiently for any given direction. In the next section we present how we approximate convex sets by computing their support function, and in the following section we extend the approach to flowpipes.

5.1.1 Approximating Convex Sets with Support Samples

A convex set can be represented by its support function, which attributes to each direction in $\mathbb{R}^n$ the signed distance of the farthest point of the set to the origin. Computing the value of the support function for a given set of directions, one obtains a polyhedron that overapproximates the set. We call this *sampling the support function*. In this chapter, we allow this computation to be approximative, i.e., a lower and an upper bound on the support function is computed. In this section, we define support samples and derive error bounds for approximations from support samples.

Because support functions are cheap, we would like to use them in our flowpipe approximation. However, in our construction it is not always possible or efficient to compute the exact value of the support function. Instead, we allow for interval bounds on the support function. Furthermore, we consider that those bounds are only computed for a finite number of directions. In the following, we examine how such bounds provide an outer approximation of the actual set and characterize the approximation error. Given a set $\mathcal{X}$, a *support sample* $r = (\ell, [r^-, r^+])$ pairs a direction $\ell \in \mathbb{R}^n$ with a real-valued interval $[r^-, r^+]$ that contains the value of the support function of $\mathcal{X}$, i.e.,

$$\rho_{\mathcal{X}}(\ell) \in [r^-, r^+]. \tag{5.2}$$

A *support sampling* is a set of support samples

$$R = \{r_1, \ldots, r_K\}, \quad \text{with} \quad r_k = (\ell_k, [r_k^-, r_k^+]).$$

Its *outer approximation* is the polyhedron

$$\lceil R \rceil = \Big\{ x \;\Big|\; \bigwedge_k \ell_k^\mathsf{T} x \le r_k^+ \Big\}, \tag{5.3}$$

i.e., given a support sampling R of $\mathcal{X}$, we have that $\mathcal{X} \subseteq \lceil R \rceil$.

From the lower bounds in the support samples we can derive a lower bound on the support function in any direction, which allows us to bound the approximation error of the outer approximation. By definition, a support sample r_k implies that there is at least one point $x \in \mathcal{X}$ such that $\ell_k^\mathsf{T} x \ge r_k^-$, as illustrated in Fig. 5.3. Let the *facet slab* of r_k be

$$\lfloor R \rfloor_k = \lceil R \rceil \cap \{\ell_k^\mathsf{T} x \ge r_k^-\}, \tag{5.4}$$

then the support function in direction ℓ cannot be lower than

$$\min\{\ell^\mathsf{T} x \mid x \in \lfloor R \rfloor_k\} = -\rho_{\lfloor R \rfloor_k}(-\ell).$$

Combining the lower bounds from all facet slabs, we obtain the following result:

Lemma 5.3. *Given a support sampling R of a nonempty compact convex set $\mathcal{X}$, the support function of $\mathcal{X}$ is bounded in any direction ℓ by $\rho_R^-(\ell) \le \rho_{\mathcal{X}}(\ell) \le \rho_R^+(\ell)$, where*

$$\rho_R^+(\ell) = \rho_{\lceil R \rceil}(\ell), \tag{5.5}$$

$$\rho_R^-(\ell) = \max_{k=1,\ldots,K} -\rho_{\lfloor R \rfloor_k}(-\ell). \tag{5.6}$$

For a given direction ℓ, the lower bound (5.6) can be reformulated as a linear program with $O(Kn)$ variables and $O(K^2)$ constraints by introducing an additional variable z:

$$\rho_R^-(\ell) = \min\Big\{ z \in \mathbb{R} \;\Big|\; \bigwedge_{k=1,\ldots,K} z \ge \ell^\mathsf{T} x_k \wedge x_k \in \lfloor R \rfloor_k \Big\}. \tag{5.7}$$

We consider two ways to measure the error between the actual set and its outer approximation: a directional error and the Hausdorff distance. The *directional error* of a support sampling R is the width of the bounds on the support function,

$$\varepsilon_R(\ell) = \rho_R^+(\ell) - \rho_R^-(\ell). \tag{5.8}$$

Let $\mathcal{B}_k = \{x \mid \|x\|_k = 1\}$ be the unit ball in the k-norm. The *directed Hausdorff distance* between sets $\mathcal{X}$,$\mathcal{Y}$ is

$$d_H(\mathcal{X}, \mathcal{Y}) = \sup_{x \in \mathcal{X}} \inf_{y \in \mathcal{Y}} \|x - y\|_2 = \inf\{\varepsilon > 0 \mid \mathcal{X} \subseteq \mathcal{Y} \oplus \varepsilon \mathcal{B}_2\},$$

and the *Hausdorff distance* is

$$d_H(\mathcal{X}, \mathcal{Y}) = \max\Big(d_H(\mathcal{X}, \mathcal{Y}), d_H(\mathcal{Y}, \mathcal{X}) \Big).$$

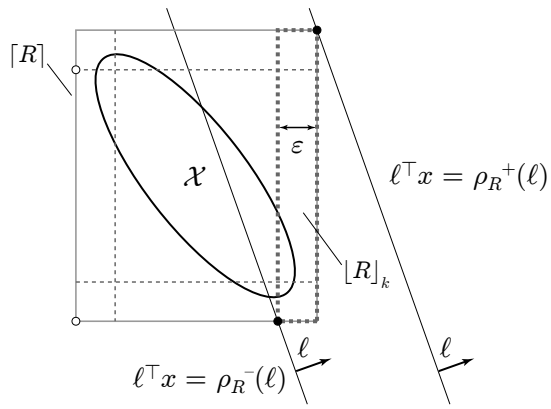


Figure 5.3: A support sampling R of a set $\mathcal{X}$ (solid black) defined over the axis directions, with lower and upper bound being ε apart. The outer approximation $\lceil R \rceil$ (solid grey) is shown together with the facet slabs $\lfloor R \rfloor_k$ (dashed), each of which contains at least one point of $\mathcal{X}$. For an arbitrary direction ℓ, the outer approximation $\lceil R \rceil$ provides an upper bound $\rho_R^+(\ell)$ on the support function, and the facet slabs a lower bound $\rho_R^-(\ell)$

Lemma 5.4. *Given a support sampling R of $\mathcal{X}$,*

$$d_H(\mathcal{X}, \lceil R \rceil) \leq \max_{\|\ell\|_2 = 1} \varepsilon_R(\ell). \tag{5.9}$$

Using the LP formulation (5.7), the above bound on the Hausdorff distance can be rewritten as a quadratic maximization problem with bilinear constraints. This bound is generally quite costly to compute. But it implies that, as more and more directions are sampled, the largest directional error tends towards a bound on the Hausdorff distance (assuming directions are uniformly distributed).

5.1.2 Approximating Flowpipes with Support Samples over Time

Our space-time construction is a natural extension of the support function representation of sets. For a given convex and bounded set of initial states $\mathcal{X}_0$, we define the flowpipe as the states reachable from this set.

Formally, let $\mathcal{X}_t$ be the states reachable from $\mathcal{X}_0$ after exactly time t,

$$\mathcal{X}_t = \{x(t) \mid x(0) \in \mathcal{X}_0, \forall \tau \in [0, t]\ \exists u(\tau) \in \mathcal{U} : \dot{x}(\tau) = Ax(\tau) + u(\tau)\}. \tag{5.10}$$

A *flowpipe segment* over a time interval $[t_1, t_2]$ is the set

$$\mathcal{X}_{t_1, t_2} = \bigcup_{t_1 \leq t \leq t_2} \mathcal{X}_t.$$

In this chapter, we assume a finite time horizon T and refer to $\mathcal{X}_{0,T}$ as the *flowpipe*. Given that $\mathcal{X}_0$ is convex and that the dynamics are affine, $\mathcal{X}_t$ is convex at any time t. For a fixed value of t, we can approximate $\mathcal{X}_t$ with a support sampling

$$R = \{(\ell_1, r_1), \ldots, (\ell_K, r_K)\},$$

where the ℓ_k given template directions, and the r_k are intervals containing the support function of $\mathcal{X}_t$. Recall that R allows us to construct an outer approximation of $\mathcal{X}_t$ and quantify the approximation error.

We describe the nonconvex flowpipe over the time interval $[0, T]$ in a similar way. Letting t vary in the time interval $[0, T]$, we consider the bounds of the interval $r_k(t) = [r_k^-(t), r_k^+(t)]$ to be continuous functions over time. For every t, $r_k(t)$ contains the support function of $\mathcal{X}_t$ in direction $\ell_k(t)$. A *flowpipe sampling* over K directions is a function F that attributes to each t a support sampling

$$F(t) = \{(\ell_1(t), r_1(t)), \ldots, (\ell_K(t), r_K(t))\}. \tag{5.11}$$

The pairs $(\ell_k(\cdot), r_k(\cdot))$ are called *flowpipe samples*. In this chapter, we consider the directions to be constant over time, and simply write ℓ_k instead of $\ell_k(t)$. By combining the outer approximation of the support sampling $F(t)$ at each time point, we obtain an outer approximation of a flowpipe segment $\mathcal{X}_{t_1,t_2}$. With Lemma 5.4 it is straightforward to derive a bound on the Hausdorff distance between the flowpipe segment and its outer approximation.

Lemma 5.5. *Let F be a flowpipe sampling (5.11) and let*

$$\lceil F \rceil_{t_1,t_2} = \bigcup_{t_1 \le t \le t_2} \lceil F(t) \rceil, \tag{5.12}$$

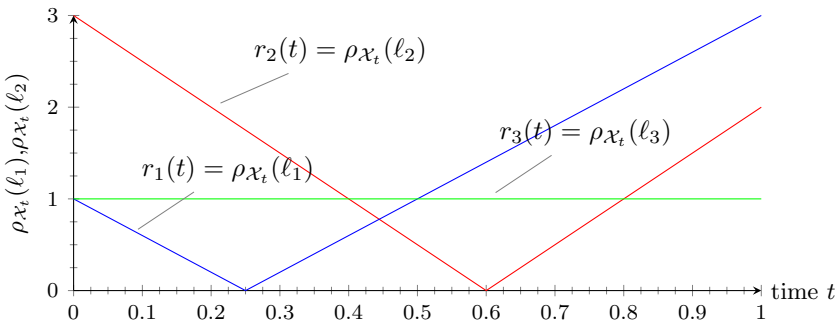
$$\varepsilon_{t_1,t_2} = \max_{t_1 \le t \le t_2} \max_{\|\ell\|_2 = 1} \varepsilon_{F(t)}(\ell). \tag{5.13}$$

Then $\mathcal{X}_{t_1,t_2} \subseteq \lceil F \rceil_{t_1,t_2}$ and the Hausdorff distance between $\lceil F \rceil_{t_1,t_2}$ and $\mathcal{X}_{t_1,t_2}$ is bounded by ε_{t_1,t_2}.

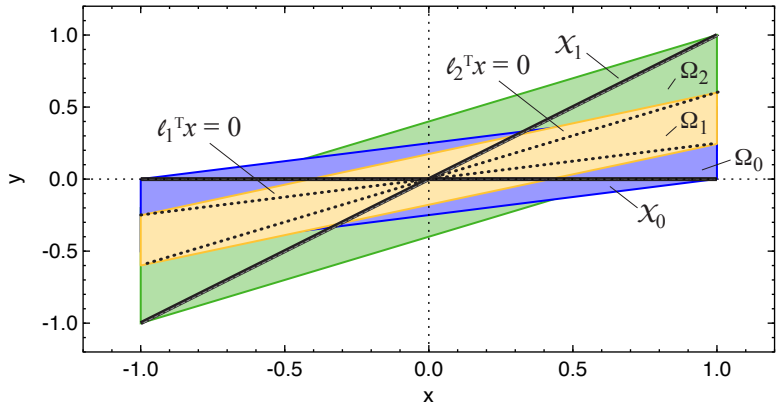
Example 5.6. *In Ex. 5.2 (hourglass), the trajectories are $x(t) = x(0), y(t) = t \cdot x(0)$, with initial states $\mathcal{X}_0 = \{-1 \le x(0) \le 1, y(0) = 0\}$. The support function over time for a direction vector $\ell = \begin{pmatrix}\alpha & \beta\end{pmatrix}$ is*

$$\begin{aligned}\rho_{\mathcal{X}_0}(\ell) &= \max_{x(0) \in \mathcal{X}_0} \begin{pmatrix}\alpha & \beta\end{pmatrix} \cdot \begin{pmatrix}x(0) & t \cdot x(0)\end{pmatrix} \\ &= \max(\alpha + \beta t, -\alpha - \beta t). \end{aligned} \tag{5.14}$$

Let's assume that flowpipe samples have been computed for directions $\ell_1 = \begin{pmatrix}-1 & 4\end{pmatrix}$, $\ell_2 = \begin{pmatrix}-3 & 5\end{pmatrix}$, $\ell_3 = \begin{pmatrix}1 & 0\end{pmatrix}$, as well as their negatives. Assuming the computation is exact, the lower and upper bounds of the flowpipes are identical. The flowpipe samples $r_1(t)$, $r_2(t)$, $r_3(t)$ are shown in Fig. 5.4(a). The resulting outer approximation $\lceil F \rceil_{0,T}$ is shown in Fig. 5.4(b) .



(a) Flowpipe samples for directions ℓ_1, ℓ_2, ℓ_3



(b) Outer flowpipe approximation $\lceil F \rceil_{0,T}$ in the state space, consisting of convex polyhedra $\Omega_0, \Omega_1, \Omega_2$

Figure 5.4: Approximating the flowpipe of Ex. 5.2 for a given set of directions ℓ_1, ℓ_2, ℓ_3, and their negatives

5.1.3 Polyhedral Approximations of Flowpipes

A flowpipe sampling describes a flowpipe in the same way that a support sampling describes a convex set, only that the flowpipe and its outer approximation can be nonconvex. We now show that the outer approximation of a flowpipe sampling with piecewise linear upper bounds is a set of convex polyhedra, one for each segment for which all upper bounds are concave.

Let F be a flowpipe sampling (5.11), such that for all k, $r_k^+(t), r_k^-(t)$ are piecewise linear. For constructing the polyhedra, we need some technical notation for describing the linear pieces. Let the i-th pieces of $r_k^+(t)$, $r_k^-(t)$ be

$$\begin{aligned} r_k^+(t) &= \alpha_{k,i}^+ t + \beta_{k,i}^+ \text{ for } \tau_{k,i}^+ \leq t \leq \tau_{k,i+1}^+, \\ r_k^-(t) &= \alpha_{k,i}^- t + \beta_{k,i}^- \text{ for } \tau_{k,i}^- \leq t \leq \tau_{k,i+1}^-. \end{aligned}$$

In the following, we consider the time interval $[t_1, t_2]$ and will assume for simplicity that $r_k^+(t)$, $r_k^-(t)$ have breakpoints at the boundary of $[t_1, t_2]$, i.e., for some $i' < i''$, $t_1 = \tau_{k,i'}^+$ and $t_2 = \tau_{k,i''}^+$. Let I_k^+, I_k^- be the sets of indices i of the

pieces of $r_k^+(t)$, respectively $r_k^-(t)$, that lie completely inside the time interval $[t_1, t_2]$. If all computed flowpipe samples are concave over $[t_1, t_2]$, their outer approximation is convex:

Lemma 5.7. *If for all k, $r_k^+(t)$ is concave on the time interval $[t_1, t_2]$, then $\lceil F \rceil_{t_1,t_2}$ is the convex polyhedron*

$$\begin{aligned} \lceil F \rceil_{t_1,t_2} &= M \, [\![F]\!]_{t_1,t_2}, \qquad \textit{where} \\ [\![F]\!]_{t_1,t_2} &= \Big\{ (x,t) \;\Big|\; t_1 \le t \le t_2 \wedge \bigwedge\nolimits_{k,i \in I_k^+} \ell_k^\mathsf{T} x \le \alpha_{k,i}^+ t + \beta_{k,i}^+ \Big\}, \end{aligned} \tag{5.15}$$

and the matrix M maps $(x,t) \in \mathbb{R}^{n+1}$ to $x \in \mathbb{R}^n$.

With Lemma 5.7, we can take a flowpipe sampling F and compute the support function of $\lceil F \rceil_{t_1,t_2}$ by solving a single LP with $O(n)$ variables and $O(KZ)$ constraints, where K is the number of template directions and Z is a bound on the number of pieces of the $r_k^+(t)$ in the time interval $[t_j, t_{j+1}]$.

With the above, we can construct a flowpipe approximation consisting of convex polyhedra $\Omega_0, \ldots, \Omega_N$ as follows:

1. Compute a piecewise linear flowpipe sample for each template direction.
2. Identify time intervals $[t_0, t_1], \ldots, [t_N, t_{N+1}]$, with $t_0 = 0$ and $t_{N+1} = T$, such that in each interval all samples have concave upper bounds.
3. Construct for each interval $[t_i, t_{i+1}]$ its convex polyhedron $\Omega_i = \lceil F \rceil_{t_i,t_{i+1}}$ using (5.15).

Example 5.8. *The flowpipe samples of Ex. 5.6, shown in Fig. 5.4(a), are all concave on the time intervals $[0, 0.25]$, $[0.25, 0.6]$, and $[0.6, 1]$. The outer approximation of the flowpipe consists of three convex polyhedra $\Omega_0 = \lceil F \rceil_{0,0.25}$, $\Omega_1 = \lceil F \rceil_{0.25,0.6}$, and $\Omega_2 = \lceil F \rceil_{0.6,1}$, shown in Fig. 5.4(b) . The facet normals of Ω_0 are ℓ_1, ℓ_3, those of Ω_2 are ℓ_2, ℓ_3, and those of Ω_3 are a linear combination of ℓ_1 and ℓ_2.*

The above approach produces a precise flowpipe approximation, but the number of polyhedra can be very large, especially if the concave intervals of the different flowpipe samples do not coincide. If an upper bound of a flowpipe samples is not concave on an interval, we can replace it by its concave envelope. The concave envelope of a piecewise linear function with N points, sorted along the time axis, can be computed in $O(N)$ with the Graham scan. The approximation error can be measured via the distance to the envelope. In Sect. 5.2, we will present a clustering technique that establishes the largest concave intervals that can be created by relaxing the upper bounds, under a desired error bound.

Note that a convex outer approximation does not imply that the flowpipe segment is convex. We now derive a bound on the approximation error by using the lower bounds of the flowpipe samples.

Lemma 5.9. *Let for all k, $r_k^+(t)$ be concave and $r_k^-(t)$ be convex on the time interval $[t_1, t_2]$. Let the k-th* facet slab *of F be*

$$[\![F]\!]_{t_1,t_2}^k = \Big\{ (x_k, t) \in [\![F]\!]_{t_1,t_2} \;\Big|\; \bigwedge_{i \in I_k^-} \ell_k^\mathsf{T} x_k \ge \alpha_{k,i}^- t + \beta_{k,i}^- \Big\}, \tag{5.16}$$

Then the Hausdorff distance between $\lceil F \rceil_{t_1,t_2}$ *and* $\mathcal{X}_{t_1,t_2}$ *is bounded by*

$$\varepsilon_{t_1,t_2} = \max_{\|\ell\|_2=1} \varepsilon_{t_1,t_2}(\ell), \qquad \textit{where} \tag{5.17}$$

$$\varepsilon_{t_1,t_2}(\ell) = \max \left\{ \ell^{\mathsf{T}} x - z \;\middle|\; (x,t) \in [\![F]\!]_{t_1,t_2} \wedge \bigwedge_{k=1,\ldots,K} z \geq \ell^{\mathsf{T}} x_k \wedge (x_k,t) \in [\![F]\!]^k_{t_1,t_2} \right\} \tag{5.18}$$

For a given direction ℓ, (5.18) can be formulated as a linear program. Consequently, Lemma 5.9, allows us to compute a bound on the directional approximation error $\varepsilon_{t_1,t_2}(\ell)$ by solving a single LP with $O(Kn)$ variables and $O(K^2Z)$ constraints. If we can solve the program for all ℓ, we obtain a bound on the Hausdorff distance of $\lceil F \rceil_{t_1,t_2}$ to the actual flowpipe segment.

5.1.4 Computing Flowpipe Samples for Affine Dynamics

We now present a way to construct flowpipe samples for affine dynamics of the form (5.1), i.e., an interval-valued function that bounds the support function of the reachable states at time t for a given direction. Our construction takes as input the initial set $\mathcal{X}_0$, a time horizon T, a template direction ℓ, and an error bound ϵ. It produces a flowpipe sample $(\ell, [r^-(t), r^+(t)])$, such that for all $0 \leq t \leq T$,

$$r^+(t) - r^-(t) \leq \epsilon.$$

The sample is piecewise quadratic and can easily be approximated by a piecewise linear sample so that the techniques of the previous section can be applied. The construction is based on the approach in [44], from which it differs in two ways: First, we include a lower bound on the support function, which is used to evaluate the approximation error at all stages including clustering. Second, instead of computing forward with a certain time-step, we start with a time step that covers the whole time horizon and recursively refine with smaller steps on subdomains where the difference between upper and lower bound exceeds the error bound.

We exploit the superposition principle to adapt the approximation separately to the autonomous dynamics (created by $\mathcal{X}_0$), and to the non-autonomous dynamics (created by $\mathcal{U}$). $\mathcal{X}_t$ can be decomposed into

$$\mathcal{X}_t = \mathcal{Z}_t \oplus \mathcal{Y}_t, \tag{5.19}$$

where $\mathcal{Z}_t = e^{At}\mathcal{X}_0$ and $\mathcal{Y}_t$ is the set of states reachable when starting from $x = 0$ instead of $\mathcal{X}_0$:

$$\mathcal{Y}_t = \{x(t) \mid x(0) = 0, \forall \tau \in [0,t]\ \exists u(\tau) \in \mathcal{U} : \dot{x}(\tau) = Ax(\tau) + u(\tau)\}. \tag{5.20}$$

Note that $\mathcal{Z}_0 = \mathcal{X}_0$ and $\mathcal{Y}_0 = 0$. We now turn to constructing a flowpipe sample $\omega(t) = [\omega^-(t), \omega^+(t)]$ for $\mathcal{Z}_t$.

Our starting point is a linear interpolation between $\mathcal{Z}_0$ and $\mathcal{Z}_\delta$. Using a forward, respectively backward, interpolation leads to error terms represented by sets $\mathcal{E}^+_\Omega$,$\mathcal{E}^-_\Omega$, as defined in Sect. 3.2.1, Lemma 3.2. The intersection of both error terms gives $\mathcal{E}_\Omega$. Using a normalized time variable $\lambda = t/\delta$, the support function of the error term $\mathcal{E}_\Omega(\delta, \lambda)$ is piecewise linear and efficient to compute.

An upper bound on the support function of $\mathcal{Z}_t$ over a time interval $[0, \delta]$ is easy to derive from the linear interpolation between $\mathcal{Z}_0$, $\mathcal{Z}_\delta$, and the above error terms. For deriving a lower bound, consider a support vector x^- of $\mathcal{Z}_0 = \mathcal{X}_0$ in direction ℓ. Since the support function of $\mathcal{Z}_\delta = e^{A\delta}\mathcal{X}_0$ is the maximum of $\ell^\mathsf{T} x$ over all $x \in \mathcal{Z}_\delta$, it is bounded below by the image of x^- at time δ, i.e., by $\ell^\mathsf{T} e^{A\delta} x^-$. From the linear interpolation between x^- and $e^{A\delta}x^-$ we derive a lower bound by subtracting a suitable error term. A similar argument can be made with the support vector x^+ at the end of the interval, and we take the maximum of both lower bounds. Using the above error terms we obtain a flowpipe sample as follows.

Lemma 5.10. *We consider the time interval $[t_i, t_{i+1}]$. Let $\delta_i = t_{i+1} - t_i$, $\ell_i = e^{At_i}{}^\mathsf{T}\ell$, $\ell_{i+1} = e^{A\delta_i}{}^\mathsf{T}\ell_i$, let x^- be a support vector of $\rho_{\mathcal{X}_0}(\ell_i)$, and x^+ be a support vector of $\rho_{\mathcal{X}_0}(\ell_{i+1})$. Let $\lambda = (t - t_i)/\delta_i$ and*

$$\omega^+(t) = (1-\lambda)\rho_{\mathcal{X}_0}(\ell_i) + \lambda\rho_{\mathcal{X}_0}(\ell_{i+1}) + \rho_{\mathcal{E}_\Omega(\delta_i,\lambda)}(\ell_i) \tag{5.21}$$

$$\begin{aligned}\omega^-(t) = &\max\{(1-\lambda)\ell_i^\mathsf{T} x^- + \lambda\ell_{i+1}^\mathsf{T} x^-,\\ &\qquad (1-\lambda)\ell_i^\mathsf{T} x^+ + \lambda\ell_{i+1}^\mathsf{T} x^+\} - \rho_{\mathcal{E}_\Omega(\delta_i,\lambda)}(\ell_i).\end{aligned} \tag{5.22}$$

Then $\omega^-(t) \le \rho_{\mathcal{Z}_t}(\ell) \le \omega^+(t)$ for all $t_i \le t \le t_{i+\delta_i}$.

The approximation error of $\omega^-(t), \omega^+(t)$ in the time interval $[t_i, t_i + \delta_i]$ is

$$\varepsilon_\omega(t_i, t_{i+1}) = \max_{t_i \le t \le t_{i+1}} \omega^+(t) - \omega^-(t). \tag{5.23}$$

Note that the approximation error decreases at least linearly with δ_i. To meet the given error bound ϵ_ω, we construct $\omega^-(t), \omega^+(t)$ and the corresponding sequence of time points t_i by establishing a list of suitable intervals. We begin with a single interval $[t_0, t_1] = [0, T]$, which covers the entire time horizon. Each interval $[t_i, t_{i+1}]$ in the list is processed in the following steps:

1. Construct $\omega^-(t), \omega^+(t)$ on the interval $[t_i, t_{i+1}]$ and compute $\varepsilon_\omega(t_i, t_{i+1})$.
2. If $\varepsilon_\omega(t_i, t_{i+1}) > \epsilon_\omega$, split the interval in two. Let $t' = (t_i + t_{i+1})/2$. Replace $[t_i, t_{i+1}]$ with intervals $[t_i, t']$, $[t', t_{i+1}]$, and process each starting with step 1.

Example 5.11. *Consider computing a flowpipe sample of Ex. 5.2 (hourglass) for direction ℓ_2 and up to an error bound of $\epsilon = 1$, as illustrated by Fig. 5.5. There are no inputs, so $r_2(t) = \omega(t)$. We start with the interval $[0, 1]$, which yields as upper bound the linear interpolation between $\omega^+(0) = 3$ and $\omega^+(1) = 2$, shown dashed in Fig. 5.5. In this example, the lower bound $\omega^-(t)$ happens to coincide with $r_2(t)$. The inital approximation error is $\varepsilon_\omega(0, 1) = 2.4$. Since this exceeds ϵ, the interval is split into two pieces, $[0, 0.5]$ and $[0.5, 1]$. The approximation errors are $\varepsilon_\omega(0, 0.5) = 0$ and $\varepsilon_\omega(0.5, 1) = 0.6$. They satisfy the error bound ϵ, and we obtain the upper bound $\omega^+(t)$ shown in Fig. 5.5.*

We now establish a flowpipe sample $\psi(t) = [\psi^-(t), \psi^+(t)]$ for $\mathcal{Y}_t$. Computing $\psi(t)$ is more difficult than computing $\omega(t)$ because there is no analytic solution for the integral over the input $u(t)$. Because of the integration, the approximation error accumulates over time. In order to guarantee that for all t

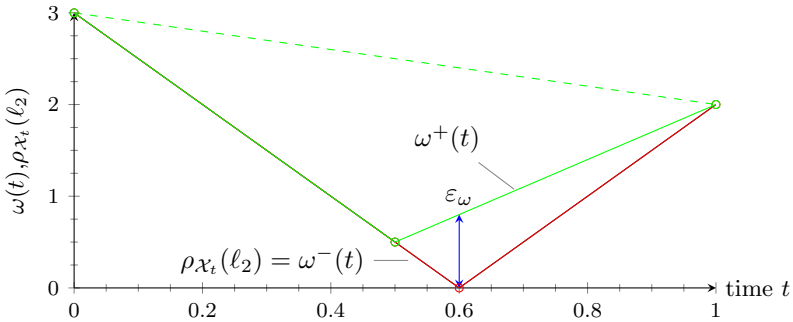


Figure 5.5: Computing a flowpipe sample of Ex. 5.2 for direction ℓ_2

the (accumulated) error of $\psi(t)$ is below a given bound ϵ_ψ, we impose that the accumulated error at the end of each interval $[t_i, t_{i+1}]$ must lie below the *error rate* $\epsilon_\psi \cdot t_{i+1}/T$. We use a two-step process: We first compute $\psi(t_i)$ at discrete points in time t_i such that the desired error rate is met. Based on these values we then define $\psi(t)$ over continuous time. To bound the approximation error we use the error term

$$\mathcal{E}_\Psi(\mathcal{U}, \delta) = \boxdot\left(\Phi_2(|A|, \delta) \boxdot (A\mathcal{U})\right).$$

For the following two lemmas, we assume the sequence of time points t_i as given. Its construction is presented afterwards, when the required error terms have been defined. We have the following bounds on $\rho_{\mathcal{Y}_t}(\ell)$ at the discrete time points t_i.

Lemma 5.12. *[44] Let $t_0 = 0, t_1, t_2, \ldots, t_N = T$ be an increasing sequence of time points. Let $\delta_i = t_{i+1} - t_i$, $\ell_i = e^{At_i\mathsf{T}}\ell$, $\ell_{i+1} = e^{A\delta_i\mathsf{T}}\ell_i$, $\psi^+_{t_0} = 0, \psi^-_{t_0} = 0$, and*

$$\psi^+_{t_{i+1}} = \psi^+_{t_i} + \delta_i \rho_{\mathcal{U}}(\ell_i) + \rho_{\mathcal{E}_\Psi(\mathcal{U},\delta_i)}(\ell_i) \tag{5.24}$$

$$\psi^-_{t_{i+1}} = \psi^-_{t_i} + \delta_i \rho_{\mathcal{U}}(\ell_i) - \rho_{-A\Phi_2(A,\delta_i)\mathcal{U}}(\ell_i). \tag{5.25}$$

Then for all t_i, $\psi^-_{t_i} \leq \rho_{\mathcal{Y}_{t_i}}(\ell) \leq \psi^+_{t_i}$.

Based on the bounds on $\rho_{\mathcal{Y}_t}(\ell)$ at the discrete times t_i, we obtain the following bounds over the intervals $[t_i, t_{i+1}]$.

Lemma 5.13. *Let $\lambda = (t - t_i)/\delta_i$ and*

$$\psi^+(t) = \psi^+_{t_i} + \lambda\delta_i\rho_{\mathcal{U}}(\ell_i) + \lambda^2\rho_{\mathcal{E}_\Psi(\mathcal{U},\delta_i)}(\ell_i), \tag{5.26}$$

$$\psi^-(t) = \psi^-_{t_i} + \lambda\delta_i\rho_{\mathcal{U}}(\ell_i) - \lambda\rho_{-A\Phi_2(A,\delta_i)\mathcal{U}}(\ell_i) - \lambda^2\rho_{\mathcal{E}_\Psi(\mathcal{U},\delta_i)}(\ell_i). \tag{5.27}$$

Then $\psi^-(t) \leq \rho_{\mathcal{Y}_t}(\ell) \leq \psi^+(t)$ for all $t_i \leq t \leq t_{i+\delta_i}$.

We construct the time intervals $[t_i, t_{i+1}]$ by refinement until the error $\psi(t)$ falls below ϵ_ψ. According to the above Lemmas, the error bound on the interval

$[t_i, t_{i+1}]$ is defined by the following sequence, starting with $\varepsilon_\psi(t_0) = 0$:

$$\begin{aligned}\varepsilon_\psi(t_i, t_{i+1}) =& \varepsilon_\psi(t_i) + \max_{0\le\lambda\le 1} 2\lambda^2 \rho_{\mathcal{E}_\Psi(\mathcal{U},\delta_i)}(\ell_i) \\ &+ \lambda\rho_{-A\Phi_2(A,\delta_i)\mathcal{U}}(\ell_i), \qquad \text{where} \qquad (5.28)\\ \varepsilon_\psi(t_{i+1}) =& \varepsilon_\psi(t_i) + \rho_{\mathcal{E}_\Psi(\mathcal{U},\delta_i)}(\ell_i) + \rho_{-A\Phi_2(A,\delta_i)\mathcal{U}}(\ell_i). \qquad (5.29)\end{aligned}$$

When choosing the time points t_i, we must ensure that largest error in the interval $[t_i, t_{i+1}]$ lies below the error bound, i.e., $\varepsilon_\psi(t_i, t_{i+1}) \le \epsilon_\psi$. To take into account that the error $\varepsilon_\psi(t_i)$ accumulates, we also ensure that the rate of the accumulated error stays below ϵ_ψ/T, i.e., $\varepsilon_\psi(t_{i+1}) \le \epsilon_\psi \cdot t_{i+1}/T$. Beginning with a single interval $[t_0, t_1] = [0, T]$, each interval $[t_i, t_{i+1}]$ is processed in the following steps:

1. Compute $\varepsilon_\psi(t_{i+1})$ and $\varepsilon_\psi(t_i, t_{i+1})$.
2. If $\varepsilon_\psi(t_{i+1}) > \epsilon_\psi \cdot t_{i+1}/T$ or $\varepsilon_\psi(t_i, t_{i+1}) > \epsilon_\psi$, split the interval in two. Let $t' = (t_i + t_{i+1})/2$. Replace $[t_i, t_{i+1}]$ with intervals $[t_i, t']$, $[t', t_{i+1}]$, and process each starting with step 1.

Using the superposition principle (5.19), we finally combine $\omega(t)$ and $\psi(t)$ to obtain a flowpipe sample for $\mathcal{X}_t$ as

$$r(t) = [r^-(t), r^+(t)] = \omega(t) + \psi(t).$$

The error bound on $r(t)$ is below $\epsilon = \epsilon_\omega + \epsilon_\psi$.

5.2 Clustering in Space-Time

Given a flowpipe sampling, our goal is to construct a sequence of convex sets that cover the flowpipe and that are no further than a given distance ϵ from it. For computational efficiency, our distance measure is the directional error in each of the sampled directions, but this implies also a distance in the Hausdorff sense.

We now given an informal description of our clustering algorithm, deferring a formal discussion to the subsections that follow. The algorithm takes as input a flowpipe sampling $F = \{(\ell_1, r_1(t)), \ldots, (\ell_K, r_K(t))\}$ and an error bound ϵ, and produces a flowpipe sampling F' by replacing the upper bounds $r_k^+(t)$ with a piecewise concave envelope with as few pieces as possible for the given error bound. The basic principle is to (over)approximate the upper bounds $r_i^+(t)$ of the flowpipe samples with a set of piecewise concave hulls $y_i(t)$, which are constructed such that they are concave over the same pieces. Recalling from in Sect. 5.1.3 that an outer approximation in the form of a convex polyhedron can be constructed for each concave piece, this effectively reduces the number of convex sets.

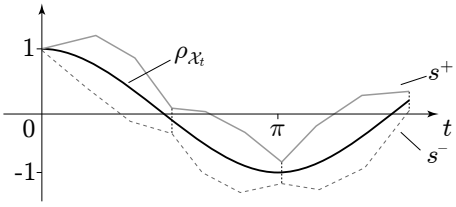
Let $\rho_i(t) = \rho_{\mathcal{X}_t}(\ell_i)$ be the actual value of the support function over time. By definition, $r_i^-(t) \le \rho_i(t) \le r_i^+(t)$. The goal of our clustering is to produce a piecewise concave hull $y_i(t)$ that is no farther than ϵ away from the actual value $\rho_i(t)$, i.e., such that $\rho_i(t) \le y_i(t) \le \rho_i(t) + \epsilon$. Since only $r_i^-(t)$ and $r_i^+(t)$ are known, we must construct the $y_i(t)$ such that

$$r_i^+(t) \le y_i(t) \le r_i^-(t) + \epsilon. \qquad (5.30)$$

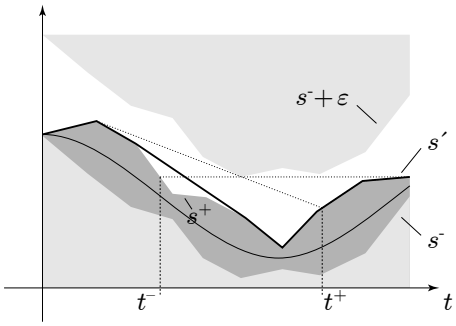
Finding the minimal number of concave pieces for a function between a lower and an upper bound is possible by establishing the *inflection intervals* of $(r_i^+(t), r_i^-(t)+\epsilon)$. The set of inflection intervals has the following property: Any piecewise concave function between $r_i^+(t)$ and $r_i^-(t)+\epsilon$ has at least one inflection point inside every inflection interval. The number of inflection intervals is thus equal to the minimum number of concave pieces of any $y_i(t)$. To have the minimum number of concave pieces, we must find the minimum number of points such that there is at least one point in every inflection interval of every sample. This turns out to be a graph coloring problem. The clustering step itself terminates with the construction of a piecewise concave hull of the flowpipe samples with a minimum number of pieces. Convex polyhedra can be derived from the concave pieces as previously described in Sect. 5.1.3.

If the clustering results in a number of concave pieces that is still considered too high, one can try to reduce the number further by recomputing the flowpipe samples with higher accuracy. This brings the $r_i^-(t)$ and $r_i^+(t)$ closer together, which increases the slack in the bounds (5.30) used for clustering. As illustrated by Fig. 5.6, the new bounds may be wide enough to admit a fewer pieces. We can obtain a lower bound on the number of pieces by computing the number of inflection intervals for the bounds

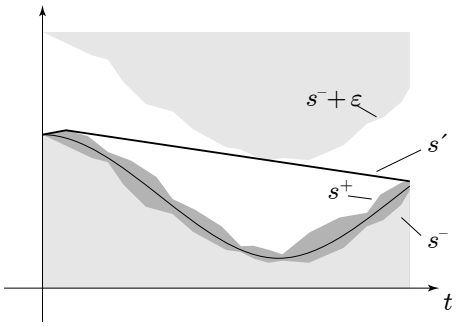
$$r_i^-(t) \leq y_i(t) \leq r_i^+(t) + \epsilon. \tag{5.31}$$



(a) Let $\mathcal{X}_t$ be a unit circle in the x/y-plane, with $\rho_{\mathcal{X}_t}(\ell) = cos(t)$ if ℓ is the x-direction. A flowpipe sample $(\ell, s^-(t), s^+(t))$ encloses this function



(b) Any piecewise linear s' between $s^+(t)$ and $s^-(t) + \varepsilon$ has at least two concave pieces, separated by an inflection point between t^- and t^+



(c) Refining s^-, s^+ leaves more slack between s^+ and $s^- + \varepsilon$, and enables an approximation s' with a single concave piece

Figure 5.6: Given a flowpipe sample with bounds $s^-(t), s^+(t)$ and a desired approximation error ε, we construct a piecewise concave function $s'(t)$ that lies between $s^+(t)$ and $s^-(t) + \varepsilon$. Fewer concave pieces in $s'(t)$ mean fewer convex sets produced by the clustering

Inflection Intervals of a Flowpipe Sample Let $l(t)$, $u(t)$ be a pair of piecewise linear functions with domain $[0, T]$ such that $l(t) \leq u(t)$. An *inflection interval* is an interval over t that contains at least one inflection point of any piecewise concave function $y(t)$ lying on or between $l(t)$ and $u(t)$, and no points that are not inflection points of a piecewise concave $y(t)$ with a minimum number of pieces. As a consequence of this definition, the minimum number of pieces of any $y(t)$ is equal to the number of inflection intervals of $(l(t), u(t))$.

Let the breakpoints of $l(t)$ be l_0 to l_N. We propose the following algorithm for finding inflection intervals (for simplicity we omit some special cases), see Fig. 5.7 for an illustration:

1. Perform a greedy piecewise concave minimal function construction from l_0 to l_N: choose at each step the point on the lower bound farthest towards l_N that is still visible.
2. Denote the breakpoints where the function is convex by $b_0, \ldots, b_z$.
3. Perform a greedy piecewise concave minimal function construction in reversed direction, from l_N to l_0.
4. Denote the breakpoints where the function is convex by $a_z, \ldots, a_0$.
5. Return the inflection intervals $I_0 = [a_0, b_0], \ldots, I_z = [a_z, b_z]$.

Proposition 5.14. *Given intervals $I_0, \ldots, I_z$ returned by the above algorithm, there exists a piecewise concave function between $l(t)$ and $u(t)$ with $z+1$ inflection points, one in every interval I_i. There exists no piecewise concave function between $l(t)$ and $u(t)$ with less than $z+1$ inflection points.*

Proof. Let us consider one of the b_i, let us denote it b for simplicity, and l_b the previous vertex in the piecewise concave piecewise linear function. l_b is on $l(t)$ (but not necessarily one of the l_i) otherwise the function would not be minimal. b is on a segment $]l_i, l_{i+1}[$ and there is a point $u' \in u(t)$ on the segment $[l_b, b]$ otherwise the function would not have been greedily constructed. Let us take x in $]b, l_{i+1}[$, any concave function on $[b, x]$ above $l(t)$ and below $u(t)$ must be above l_b and x and below u' which is not possible since u' is below $[l_b, x]$. Thus any piecewise concave function must contain at least one inflection point on $]l_b, x[$, and thus at least one on $]l_b, b]$, and one on each $]l_{b_i}, b_i]$. Since the intervals are disjoint, the greedy algorithm reaches a minimum number of inflexion point. Similarly any piecewise concave function must contain at least one inflection point on each $[a_i, l_{a_i}[$. □

Combining Inflection Intervals Having established the inflection intervals for each flowpipe sample, we combine them to find the minimum number of inflection points, as well as their possible positions, for our piecewise cover of all samples. Recall that the pieces of the piecewise cover we seek are common to all samples. We therefore need to pick at least one point from every inflection interval of every sample. To minimize their number, we construct their common sub-intervals, which we call *overlap intervals*.

For each function we have a (possibly empty) set of inflection intervals I_i obtained using the algorithm of Prop. 5.14. For the following it is not relevant that the inflection intervals originate from different functions, so let $I_0, I_1, \ldots, I_z$

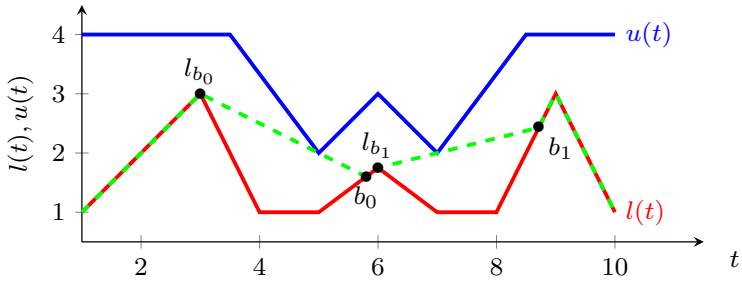
simply be the set of all inflection intervals. We need to partition the intervals into groups inside which all intervals overlap. The output of the algorithm consists of the groups and for each group their common *overlap intervals* J_j.

Finding maximal groups of overlapping intervals is equivalent to a coloring problem. Each color j corresponds to one of the groups and defines an overlap interval, which consists of the overlap between all members of the group. Two intervals I_1, I_2 need to be colored differently if they do not overlap, i.e., if $I_1^+ < I_2^-$ or $I_2^+ < I_1^-$. This relationship is captured by the *comparability graph*, whose vertices are the intervals I_i. Its edges are given by $I_i \to I_j \Leftrightarrow I_i^+ < I_j^-$, which is the so-called interval ordering (a strict partial order). Our problem is equivalent to finding a coloring of the comparability graph with the smallest number of colors such that no two adjacent vertices have the same color. Once the intervals have been colored, each color corresponds to a set of intervals that all overlap. We may freely choose an inflection point from inside the common region for that color.

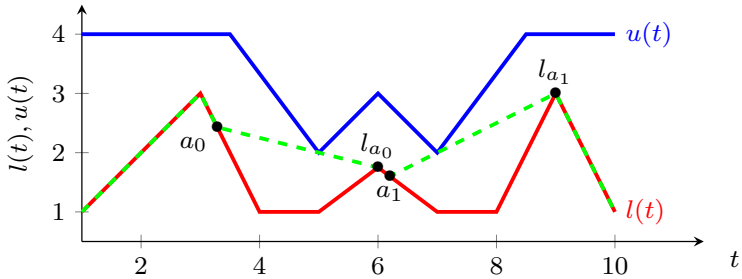
It is known that the interval ordering is a perfect elimination ordering of the comparability graph of a set of intervals. Consequently, a greedy coloring algorithm produces the optimal result if it chooses the vertices in an order that satisfies the interval ordering [69]. Such an order of the vertices can be obtained by a topological sort, i.e., a depth-first search in the graph. The total complexity is determined by the size of the comparability graph and therefore $O(z^2)$, where z is the number of intervals.

Example 5.15. *A set of intervals is shown in Fig. 5.8(a), as well as the cut intervals. The corresponding DAG is shown in Fig. 5.8(b). A depth-first search on the graph (going from the top down, left to right) yields the total order* de, fg, ab, bc, gh. *Greedy coloring in this order yields the color* 1 *for* ab, de, fg *and* 2 *for* bc, gh. *The extracted intervals* J_1 *and* J_2 *are shown in Fig. 5.8(a).*

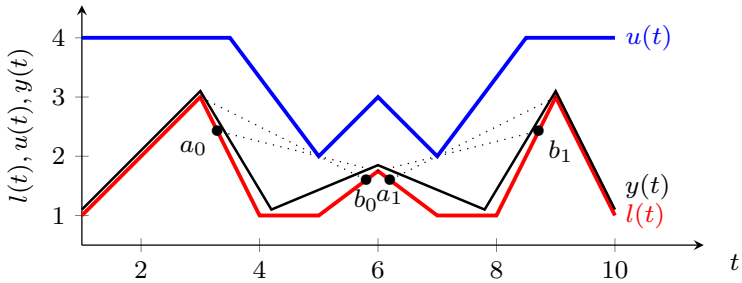
Choosing Inflection Points Our final set of inflection points consists of one point from each overlap interval J_j. The approximation error of this choice can be measured as the distance of the lower bounds to the resulting piecewise concave functions. The choice in one inflection interval generally influences the approximation error of its neighboring intervals as well, so the optimal choice is a multivariate optimization problem. In our experiments, we have observed that the overlap between the intervals of different directions is usually small, and that choosing inflection points in the middle of each interval yields results that are close to the local optimum.



(a) Greedy scan of visible lower bound points from l_0

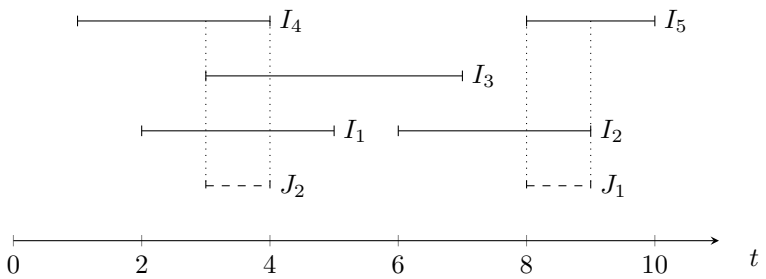


(b) Greedy scan of visible lower bound points from l_N

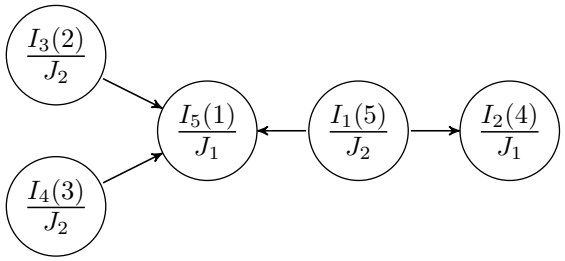


(c) The greedy scans define inflection intervals $I_0 = [a_0, b_0], I_1 = [a_1, b_1]$; for illustration, we show a piecewise concave function $y(t)$ with the minimum number of pieces

Figure 5.7: Finding the set of inflection intervals. Any piecewise concave function above $l(t)$ and below $u(t)$ has at least one inflection point inside each inflection interval



(a) Inflection intervals $I_1, \ldots, I_5$ and overlap intervals J_1, J_2



(b) Comparability graph, showing for each node I_i in parentheses its order from a topological sort and its overlap interval J_j (color), obtained by greedy coloring in that order

Figure 5.8: Finding the smallest set of inflection points in a set of overlapping inflection intervals can be solved by greedy coloring of the comparability graph in any order conform to the interval ordering

5.3 Experimental Results

In this section, we experimentally compare the proposed algorithms for flowpipe approximation and clustering with the approximation from Chapter 3. An implementation of these algorithms is used in the verification tool SpaceEx [44] to compute an overapproximation of the reachable states of a *hybrid* system. Recall that computing the reachable states of a hybrid systems involves computing the successor states of time elapse (flowpipe approximation) and the successor states of discrete transitions. The latter involves intersecting the flowpipe with the invariants of source and target locations, as well as the transition guard, which can be carried out efficiently on polyhedra. This motivates why we consider the final result of the flowpipe approximation to be the polyhedral outer approximation as described in Sect. 5.1.3. Note that other variants of the reachability algorithm avoid polyhedra, e.g., by carrying out the intersection on the support function level through transformation into an optimiziation problem [66, 46]. We compare the following variants:

LGG (state-space approximation without clustering) variable time-step flowpipe construction in the state-space, then outer polyhedral approximation, both as in Chapter 3,

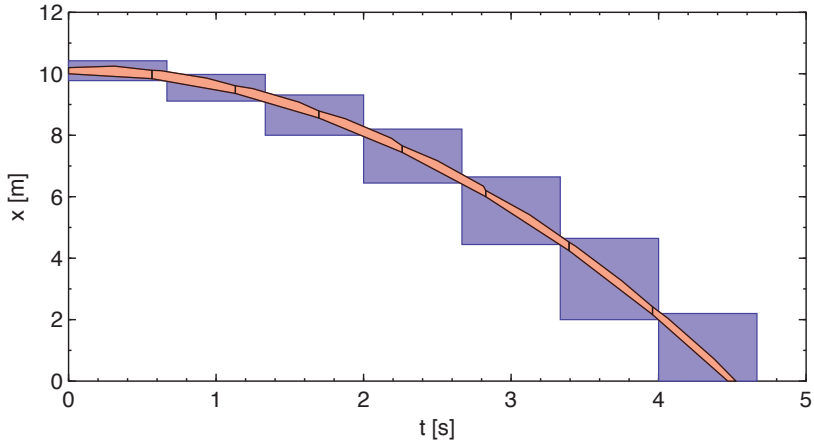
STA (space-time approximation with all pieces) flowpipe construction as in Sect. 5.1, then outer polyhedral approximation of all pieces as in Sect. 5.1.3 (no clustering),

STC (space-time approximation with clustering) flowpipe construction as in Sect. 5.1 and clustering as in Sect. 5.2, then outer polyhedral approximation as in Sect. 5.1.3.

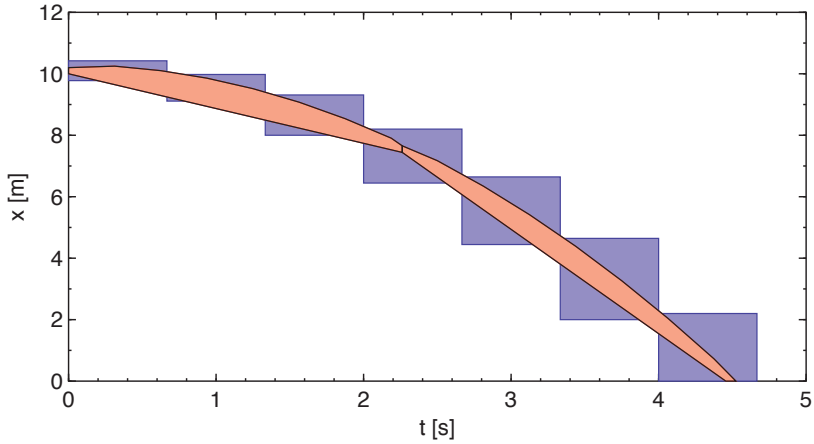
Note that the STA/STC implementation is still a prototype, and we expect that memory consumption and clustering runtime can be reduced. The parameter settings are not entirely comparable between LGG and STA/STC, since the error bounds in STA/STC are conservative, while in LGG they are mere estimates that do not take the nonconvexity error in account. The error bound in STC measures the total error, including both flowpipe approximation and clustering. We choose that 80% of the error can be taken up by the flowpipe approximation, so that at least 20% of the error bound remain as slack for the clustering step.

To avoid a lengthy description of the models, they are available for download on the SpaceEx website [37]. For illustration, consider a ball in free-fall together with a clock, with 3 variables x, v, t, dynamics $\dot{x} = v$, $\dot{v} = -1$, $\dot{t} = 1$, and initial states $10 \leq x \leq 10.2$, $v = t = 0$. We construct the flowpipe until x falls below 0. The axis directions are used as template directions, so LGG creates bounding boxes of flowpipe segments in the state space. The flowpipe approximation of STA/STC creates a bounding box for each point in time, which projected to the state space yields polyhedra with facet normals other than the template directions, as Fig. 5.9 illustrates. The error bound $\epsilon = 1$ is fairly large, and the clustering step in STC uses the slack to reduce the number of sets from 8 sets (STA) to 2 sets.

Table 5.1 shows performance results obtained on a laptop with i7 processor and 8 GB RAM. All examples use the axis directions as template directions. For each algorithm, the table shows the time taken for flowpipe approximation, the



(a) LGG (blue) versus STA (red)



(b) LGG (blue) versus STC (red)

Figure 5.9: Flowpipe approximation of a ball in free fall (position over time), with axis directions as template directions and a directional error bound $\epsilon = 1$

time taken for clustering and constructing the polyhedral approximation, and the memory consumption. As an implementation-independent indicator of the computational cost, it shows the total number of times the support function of the initial set has been evaluated. The key column is the total number of convex sets covering the flowpipe, and the goal of the STC algorithm is to reduce it as much as possible for the given error bound.

The results indicate that the flowpipe approximation in space-time with clustering (STC) can produce a flowpipe cover with a small number of sets, while meeting the desired directional error bounds. The construction uses template directions, with which the reachable set is approximated in space-time, pointwise for every time instant. The projection onto the state space produces polyhedra with facet normals that are linear combinations of the template directions. Compared to our previous work, which approximates the flowpipe directly in

Table 5.1: Performance results for different examples

Algo	*Fl.T* [s]	*Cl.T.* [s]	Mem. [MB]	#eval	#sets
Helicopter with controller & clock, 29 variables, $\epsilon = 0.025$					
LGG	5.9	–	14	85144	1440
STA	9.0	3.6	2390	103726	2649
STC	9.4	4.8	203	103726	32
Helicopter with diff. controller & clock, 29 variables, $\epsilon = 0.025$					
LGG	10.4	–	15	150278	2563
STA	14.5	0.0	2620	154026	2568
STC	14.3	19.0	225	154026	10
Ball in free-fall & clock, 3 variables, $\epsilon = 0.001$					
LGG	0.04	–	11	1770	261
STA	0.02	0	14	1453	85
STC	0.02	0	12	1453	56
Ball i. ff. w. input disturbance & clock, 3 variables, $\epsilon = 0.001$					
LGG	6.3	–	94	344676	17208
STA	29.9	0	15	1580834	256
STC	29.9	0	12	1580834	64
Three-tank system, 3 variables, $\epsilon = 0.125$					
LGG	0.03	–	2	1032	105
STA	0.03	0	15	756	137
STC	0.03	0	2	756	9
Overhead crane, 4 variables, $\epsilon = 0.05$					
LGG	0.12	–	11	3944	369
STA	0.17	0	37	4896	633
STC	0.17	0	13	4896	15

Fl.T: flowpipe construction time, *Cl.T.*: clustering and polyhedral approximation time, Mem.: memory consumption, #eval: number of evaluations of the support function of the initial set, #sets: number of convex sets covering the flowpipe

the state space, this can improve precision and reduce the number of sets at the same time.

Chapter 6

Property-Based Template Refinement

In our experiences with applying scalable flowpipe approximation algorithms, the number of continuous sets that are produced (more accurately, the number of symbolic states) tends to grow quickly and become a limiting factor. Clustering is usually applied to help reduce this number, and optimal clustering can be carried out with support functions as done in Chapter 5. This number of sets is aggravated dramatically by spurious transitions, i.e., transitions that are enabled as an artifact of the overapproximation. The approximation accuracy can be improved by reducing time steps and increasing the number of directions, but if done indiscriminately this leads to large computational cost: to guarantee a Hausdorff error of ε in n dimensions, the support function must be evaluated $O(1/\varepsilon^{n-1})$ times [68].

We propose a procedure to show that a transition is spurious, i.e., its guard set is unreachable. It aims at using as few directions as possible, and adjusting the accuracy automatically. We call this separating the guard set from the flowpipe (as opposed to safety), in order to differentiate it from showing safety over all runs of the hybrid automaton. Our approach is based on the separation of two convex sets: efficient algorithms are known that produce a hyperplane separating the two sets, and its normal vector is a suitable template direction for the support function algorithm.

We propose two different ways to turn the flowpipe separation problem into a sequence of convex separation problems. In a *convexification-based* approach, we approximate the flowpipe with a finite number of convex set as in [42]. To each of these sets, we apply the above convex separation algorithm. In a *pointwise* approach, we run the convex separation algorithm at discrete points in time. The result (separation or overlap) is propagated along the time axis using continuous-time bounds on the support function of the flowpipe computed as in [42]. The main contributions of this chapter are as follows:

- We propose a novel construction of inner approximations of convex sets based solely on support functions (not support vectors). This construction is sound even for approximate computations. (Sect. 6.1)

- We propose a novel procedure for separating convex sets using only approximately computed values of support functions. To the best of our knowledge, this is the first such procedure using only support functions (not support vectors) and the first that is sound even for approximate computations. (Sect. 6.2.1)
- For the *point-wise* approach, we incorporate both *static directions*, where we check for how long the same hyperplane (possibly shifted) still separates the flowpipe (Sect. 6.3.2), and *dynamic directions*, where we rotate the separating hyperplane with the adjunct dynamics of the system (Sect. 6.3.2). These methods are complimentary since there are systems where either one or the other technique, but not both, can show separation over an infinite time horizon.

The problem of showing that a given "unsafe" set (in our case, the guard set) is not reachable is known as the safety problem. Various approaches exist, and due to lack of space we cite only a small selection. In [7], predicate abstractions are used to refute counter-examples of hybrid systems. The separating hyperplanes that we construct can be viewed as such predicates, although in our setting they need only be satisfied over intervals of time. In [34], abstractions based on eigenforms are refined using counter examples until safety is shown. However the approach is limited to deterministic dynamics, while we can handle additive nondeterminism in the ODEs. Alternating forward and backward reachability between the initial and the unsafe set can be used to show safety, but there are inherent problems with numerical accuracy, since a stable system becomes unstable when going backwards in time [76]. The main difference to all these approaches is that we are only looking for a technique to detect as quickly as possible when a set is unreachable within a location; the goal is not to decide the safety problem.

The remainder of this chapter is organized as follows. In the next section, we present approximate support functions, which we use to represent convex sets that can be only computed approximately. In Sect. 6.2, we present our algorithms for approximating and separating convex sets based on approximately computed support functions. These algorithms are applied to flowpipe separation using convexification in Sect. 6.3.1, and using point-wise separation in Sect. 6.3.2. Experimental results are shown in Sect. 6.4.

6.1 Approximate Support Functions

In practice, we compute the support function of a set with limited accuracy, e.g., due to floating point computations or to keep the computational costs to a minimum. An *approximate support function* is a function $\mathsf{support}$ that given a direction ℓ and an accuracy $\varepsilon > 0$ produces an upper bound on the support function. We require the bound to be within ε of the true value:

$$\mathsf{support}(\mathcal{X}, \ell, \varepsilon) - \varepsilon \leq \rho_{\mathcal{X}}(\ell) \leq \mathsf{support}(\mathcal{X}, \ell, \varepsilon). \tag{6.1}$$

The above form of describing approximate support functions is intended to lead to simpler notation than the interval-valued support functions used in Chapter 5; the results essentially equivalent.

Outer Approximation: We briefly recall the outer approximation from Sect. 5.1.1. Consider a set of directions $L = \{\ell_1, \ldots, \ell_N\}$ and values $s_k^+ = \mathsf{support}(\mathcal{X}, \ell_k, \varepsilon)$ for $i = 1, \ldots, N$. This gives the *outer approximation*

$$\lceil \mathcal{X} \rceil_L = \bigcap_{k=1,\ldots,K} \{\ell_k^\mathsf{T} x \leq s_k^+\}, \tag{6.2}$$

which satisfies $\mathcal{X} \subseteq \lceil \mathcal{X} \rceil_L$. At least one point $x \in \mathcal{X}$ is inside the *facet slab* associated with ℓ_k,

$$\lfloor \mathcal{X} \rfloor_k = \lceil \mathcal{X} \rceil_L \cap \{\ell_k^\mathsf{T} x \geq s_k^+ - \varepsilon\}. \tag{6.3}$$

With Lemma 5.3, the bounds on the support function of $\mathcal{X}$ are

$$\rho_{\mathcal{X}}^-(\ell) \leq \rho_{\mathcal{X}}(\ell) \leq \rho_{\mathcal{X}}^+(\ell),$$

where

$$\rho_{\mathcal{X}}^+(\ell) = \rho_{\lceil \mathcal{X} \rceil}(\ell), \tag{6.4}$$

$$\rho_{\mathcal{X}}^-(\ell) = \max_{k=1,\ldots,N} -\rho_{\lfloor \mathcal{X} \rfloor_k}(-\ell). \tag{6.5}$$

The above bounds on the support function can be used to compute an inner approximation, i.e., a set of points that are guaranteed to be inside the set. This will be discussed next.

Inner Approximation: Once the support function of a set $\mathcal{X}$ has been evaluated a number of times, the obtained values can be used to construct facet slabs. From these facet slabs we can derive an underapproximation of $\mathcal{X}$ by using the following criterion: a point x is in $\mathcal{X}$ if

$$\forall x_1 \in \lfloor \mathcal{X} \rfloor_1, \ldots, x_N \in \lfloor \mathcal{X} \rfloor_N : x \in \mathrm{CH}(x_1, \ldots, x_N). \tag{6.6}$$

However, this set is costly to compute because it involves quantifier alternation and bilinear constraints. To obtain an underapproximation of $\mathcal{X}$ with (relatively) little cost, we first estimate a point for each facet, then construct their convex hull, and finally shrink this set sufficiently to be sure that it is an underapproximation.

To estimate a point for a facet of the approximation, we use a center. A point $x \in \mathcal{X}$ is a *Chebyshev center* of $\mathcal{X}$ if it is the center of the largest ball that lies inside $\mathcal{X}$. The set of Chebyshev centers satisfies all constraints when they are tightened by the same amount, and as long as they are satisfiable. If the polyhedron is flat, i.e., its constraints contain equalities, these centers degenerate. We therefore compute them regarding the relative interior of $\mathcal{P}$. Assuming $\mathcal{P}$ has k equalities, let $\mathcal{P} = \{\bigwedge_{i=1}^k a_i^\mathsf{T} x = b_i \wedge \bigwedge_{i=k+1}^m a_i^\mathsf{T} x \leq b_i\}$. The *relative Chebyshev center* x^* and its radius z^* are

$$< x^*, z^* >= \operatorname*{argmax}_{x,z \geq 0} z \text{ s.t. } \bigwedge_{i=1}^k a_i^\mathsf{T} x = b_i \wedge \bigwedge_{i=k+1}^m a_i^\mathsf{T} x + \|a_i\| z \leq b_i. \tag{6.7}$$

Given any points $c_1, \ldots, c_N \in \lceil \mathcal{X} \rceil_L$, we define our underapproximation by shrinking their convex hull as follows.

Proposition 6.1. *Given a set of points $c_1, \ldots, c_N \in \lceil \mathcal{X} \rceil_L$, let a_i be the normal vectors of their convex hull, i.e.,*

$$\mathrm{CH}(c_1, \ldots, c_N) = \left\{ \bigwedge_{i=1}^{M} a_i^\mathsf{T} x \leq b_i \right\}.$$

Let J_i be the indices of the points that lie on the border of the i-th constraint, i.e., $J_i = \{j \mid a_i^\mathsf{T} c_j = b_i\}$, and let

$$b_i^- = \min_{j \in J_i} -\rho_{\lfloor \mathcal{X} \rfloor_j}(-a_i).$$

Then the set $\mathcal{C}^- = \{c \mid \bigwedge_{i=1}^{M} a_i^\mathsf{T} c \leq b_i^-\}$ is a subset of $\mathcal{X}$.

6.2 Separating Convex Sets

A classic way to show that two convex sets do not overlap is to find a hyperplane that separates them (the sets lie on opposites sides of the plane). Efficient algorithms for finding a separating hyperplane are known, e.g., closest points algorithms like the *Gilbert-Johnson-Keerthi (GJK) algorithm* or the *Chung-Wang algorithm*, see [86]. We refer to these as *convex separation algorithms*. In this section, we propose convex separation algorithms that differ in two aspects:

- We consider the case where only the value of the support function can be computed, while classical methods are based on computing points in the set (support vectors).
- We take into account that the support function is computed with finite accuracy, i.e., up to an interval that contains the exact value.

The following well-known lemma expresses separation with support functions.

Lemma 6.2 (Separation of convex sets)**.**
Given two compact convex sets $\mathcal{R}, \mathcal{S}$, let $\mathcal{Q} = \mathcal{R} \oplus (-\mathcal{S})$, i.e.,$\rho_{\mathcal{Q}}(d) = \rho_{\mathcal{R}}(d) + \rho_{\mathcal{S}}(-d)$. $\mathcal{R}$ and $\mathcal{S}$ are separated if and only if $0 \notin \mathcal{Q}$, or, equivalently, there is a $d^* \in \mathbb{R}^n$ with

$$\rho_{\mathcal{Q}}(d^*) < 0. \tag{6.8}$$

If d^* exists, any hyperplane $\mathcal{H} = \{x \mid {d^*}^\mathsf{T} x = b\}$ with b in the open interval $(\rho_{\mathcal{R}}(d^*), -\rho_{\mathcal{S}}(-d^*))$ separates $\mathcal{R}$ and $\mathcal{S}$.

In the following, we present separation algorithms adapted to approximately computing support functions.

6.2.1 Separation using Directed Approximation

We now propose a procedure for deciding the separation problem, based on iteratively constructing inner- and outer approximations of $\mathcal{Q}$. It is based on a polyhedral approximation algorithm called *Mutually Converging Polytopes* (MCP) by Kamenev [68], which approximates a convex set with the asymptotically optimal number of evaluations of the support function.

Given $\mathcal{Q}$ and a given number of iterations $k_{\max}$, the MCP algorithm constructs an outer approximation Q_k with at most k facets and an inner approximation C_k with at most k vertices as follows:

1. Start with $n+1$ affinely independent directions d_i. In each direction d_i, compute the support vector c_i of $\mathcal{Q}$. Let $k := n+1$.
2. Compute the outer approx. $Q_k := \bigcap_{i=1}^{k} \{d_i^\mathsf{T} x \leq d_i^\mathsf{T} c_i\}$.
3. Compute the inner approx. $C_k := \mathrm{CH}(c_1, \ldots, c_k)$ in constraint representation, and let L be its set of constraints.
4. For each constraint $a_i^\mathsf{T} x \leq b_i$ in L, compute the directional distance δ_i between the inner and the outer approximation, $\delta_i := (\rho_{\mathcal{Q}_k}(a_i) - b_i)/||a_i||$. Let $d_{k+1} := a_{i_{\max}}$ with $i_{\max} = \mathrm{argmax}_i\, \delta_i$.
5. Compute the support vector in the new direction d_{k+1}.
6. If $k = k_{\max}$, stop. Otherwise, let $k := k+1$ and go to step 2.

The MCP algorithm has optimal convergence rate, see [68] for details. The Haussdorff distance between the outer and inner approximation is bounded by the value of $\delta_{i_{\max}}$, and converges to 0; in this sense, the algorithm is *complete*.

The main steps of the MCP algorithm are inherited by our algorithm, but it differs in three important ways:

- Instead of support vectors, we use an inner *estimation*, i.e., points which might not actually be in $\mathcal{Q}$. This makes the algorithm applicable to using only support function values and to approximate computations.
- The inner *estimation* is used for choosing the next direction, while the inner *approximation* (points which are known to be in $\mathcal{Q}$), is used only as a termination criterion in case of overlap.
- We refine only in directions that are still necessary to decide whether $\mathcal{Q}$ contains 0.

We use the following notation: Throughout, we use the index k to indicate the iteration. Let d_k be the direction in which the approximation is refined in the k-th iteration. Let r_k^+ be a bound on the support of $\mathcal{Q}$ in direction d_i, and with accuracy ε_k, $r_k^+ = \mathsf{support}(\mathcal{Q}, d_k, \varepsilon_k)$. Let Q_k be the outer approximation, i.e.,

$$Q_k = \lceil \mathcal{Q} \rceil_{D_k} = \bigcap_{i=1}^{k} \{d_i^\mathsf{T} x \leq r_i^+\}.$$

Let $S_{k,i}$ be the facet slab of Q_k in direction d_i,

$$S_{k,i} = Q_k \cap \{d_i^T x \geq r_i^+ - \varepsilon_i\},$$

and let $c_{k,i}$ be a point in $S_{k,i}$ lying on a facet of Q_k, i.e.,

$$c_{k,i} \in S_{k,i} \cap \{d_i^T x \geq \rho_{Q_k}(d_i)\}.$$

Note that $c_{k,i}$ can be any point in $S_{k,i}$, e.g., the relative Chebyshev center. We choose them on the border of Q_k because this allows for an efficient, incremental, construction of their convex hull. Let $C_k = \mathrm{CH}(c_{k,1}, ..., c_{k,k})$ be the convex hull of the centers represented in constraint form. Let e_i be the n-dimensional vector with its i-th entry being 1 and all other entries being zero. Let $\varepsilon \geq 0$ be the accuracy used when evaluating the support function evaluation, and let $\varepsilon_{\min} \geq 0$

be a minimum accuracy that serves as termination criterion in case separation can not be decided.

Our *Directed Approximation* algorithm takes as inputs $\mathcal{Q} = \mathcal{R} \oplus (-\mathcal{S})$, an initial accuracy ε_0, a termination threshold accuracy $\varepsilon_{\min}$, and an eagerness parameter $\alpha > 1$ that represents the trade-off between sampling more directions and using a higher accuracy. The algorithm proceeds as follows:

1. Initialization: Choose as initial directions the normal vectors of a regular simplex: Let $d_i := e_i$ for $i = 1, ..., n$, and $d_{n+1} := -\sum_{i=1}^{n} e_i$. Let $k := n + 1$. Compute $r_i^+ = \mathsf{support}(\mathcal{Q}, d_i, \varepsilon_i)$ for $i = 1, \ldots, k$, with $\varepsilon_i := \varepsilon_0$.

2. Construct the outer approximation Q_k, its facet slabs $S_{k,1}, \ldots, S_{k,k}$, and points on the facets $c_{k,1}, ..., c_{k,k}$.

3. Compute the convex hull C_k in constraint representation. Decide, which constraints of C_k are relevant by measuring the directional distance between the inner approximation and zero. The constraints are contracted to obtain an inner approximation of $\mathcal{Q}$.

   (a) For each constraint $a_i^\mathsf{T} x \leq b_i$ of C_k do

      i. $J_i := \{j \mid a_i^\mathsf{T} c_j = b_i\}$. (indices of adjacent c_i)

      ii. $b_i^- := \min_{j \in J_i} -\rho_{S_{k,i}}(-a_i)$.

   (b) Let $L = \{a_i^\mathsf{T} x \leq b_i^- \mid b_i^- < 0\}$. (constraints already satisfied by $x = 0$ need not be refined)

   (c) If $L = \{\}$, stop with result "overlap"

4. Decide in which direction to refine, based on the distance δ between the relevant constraints and the outer approximation.

   (a) For each constraint $a_i^\mathsf{T} x \leq b_i$ in L, let $\delta_i := (\rho_{Q_k}(a_i) - b_i)/||a_i||$.

   (b) Let $d_{k+1} := a_{i_{\max}}$ with $i_{\max} = \operatorname{argmax}_i \delta_i$.

   (c) If $\delta_{i_{\max}} \leq \alpha\varepsilon_k$, let $\varepsilon_{k+1} := \varepsilon_k/10$, else $\varepsilon_{k+1} := \varepsilon_k$.

   (d) If $\delta_{i_{\max}} \leq \varepsilon_{\min}$, stop with result "unknown".

5. Compute an upper bound on the support function in the new direction and with maximum error ε.

   (a) $r_{k+1}^+ := \mathsf{support}(\mathcal{Q}, d_{k+1}, \varepsilon)$.

   (b) If $r^+ < 0$, stop with result "separation".

6. Let $k := k + 1$ and go to step 2.

The eagerness parameter α is motivated as follows: Even assuming that C_k converges to within distance ε_k of $\mathcal{Q}$ (we have no guarantee), we have that $\delta_{i_{\max}} \to \varepsilon_k$, which may lead to infinitely many iterations without ever satisfying $\delta_{i_{\max}} \leq \varepsilon_k$. Thus we must decrease ε_k at some point while $\delta_{i_{\max}} > \varepsilon_k$ still holds, which is guaranteed by choosing $\alpha > 1$. Larger values of α lead to a faster decrease of ε_k.

Lemma 6.3. *There result of the Directed Approximation algorithm is sound if it returns "separation" or "overlap". If it returns "unknown", the distance between $\mathcal{R}$ and $\mathcal{S}$ is bounded above by*

$$\delta = \min_x \|x\|^2 \ s.t. \ \bigcap_{i=1}^{k} \{a_i^\mathsf{T} x \leq b_i^-\}.$$

A demonstration of the directed approximation algorithm can be seen in Fig. 6.1. On the left hand side, $\mathcal{R}$ and $\mathcal{S}$ are shown. On the right hand side, the according $\mathcal{Q}_k$ and all $\mathcal{C}_k$ are shown. Fig. 6.2 shows the set of $\mathcal{C}_k$. We observe that most of the points are concentrated around the origin.

6.2.2 Adapted GJK Algorithm

Given compact convex sets $\mathcal{R}, \mathcal{S}$, a closest point algorithm computes the (not necessarily unique) pair of points $r^* \in \mathcal{R}$ and $s^* \in \mathcal{S}$ that are closest to each other. Finding such r^*, s^* can be reduced to finding the (unique) $q^* \in \mathcal{Q}$ in closest to 0. If $q^* = 0$, then $\mathcal{R}$ and $\mathcal{S}$ overlap. Otherwise, $d = q^*$ is the normal vector of a separating hyperplane as in Lemma 6.2.

The *Gilbert-Johnson-Keerthi (GJK) algorithm* finds such a q^* iteratively by computing maximizers. It takes advantage of the following property: Any $q \in \mathcal{Q}$ is closest to 0 if and only if q is the minimizer of $\mathcal{Q}$ in direction $d = q$, and this point is unique. Note that a minimizer of $\mathcal{Q}$ in direction d is a maximizer (support vector) of $\mathcal{Q}$ in direction $-d$. A rudimentary form of the algorithm goes as follows:

1. Start from an arbitrary direction d_0. Let $k = 0$.
2. Compute a point q_k that maximizes $d_k^\mathsf{T} q$ for $q \in \mathcal{Q}$.
3. Let q_k^* be the point in $\mathrm{CH}\{q_0, \ldots, q_k\}$ closest to 0, and let $d_{k+1} = -q_k^*$.
4. If $d_k^\mathsf{T} d_{k+1} = \|d_k\| \|d_{k+1}\|$, then stop. The point in $\mathcal{Q}$ closest to 0 is q_k^*.
5. Let $k \leftarrow k + 1$ and go to step 2.

The GJK algorithm is guaranteed to converge towards the closest point, and terminate if $\mathcal{Q}$ is a polytope. Note that if $0 \in \mathrm{CH}\{q_0, \ldots, q_k\}$, then $\mathcal{R}$ and $\mathcal{S}$ overlap, and the algorithm terminates with $q_k^* = 0$. The termination criterion in step 4 is usually relaxed to

$$|d_k^\mathsf{T} d_{k+1} - \|d_k\| \|d_{k+1}\|| \leq \varepsilon$$

for some given tolerance level $\varepsilon \geq 0$. If one is only interested in showing separation, the criterion (6.8) can be used to terminate early. Several efficiency improvements are known, but are omitted here for lack of space.

The GJK algorithm is not directly applicable in our setting, because we can only compute approximate support functions, not the corresponding support vectors. We now present a variation of the GJK algorithm that is solely based on approximate support functions.

Because we can not compute maximizers of $\mathcal{Q}$, we use centers of facet slabs instead. Since these points may not actually be in $\mathcal{Q}$, we must find new directions

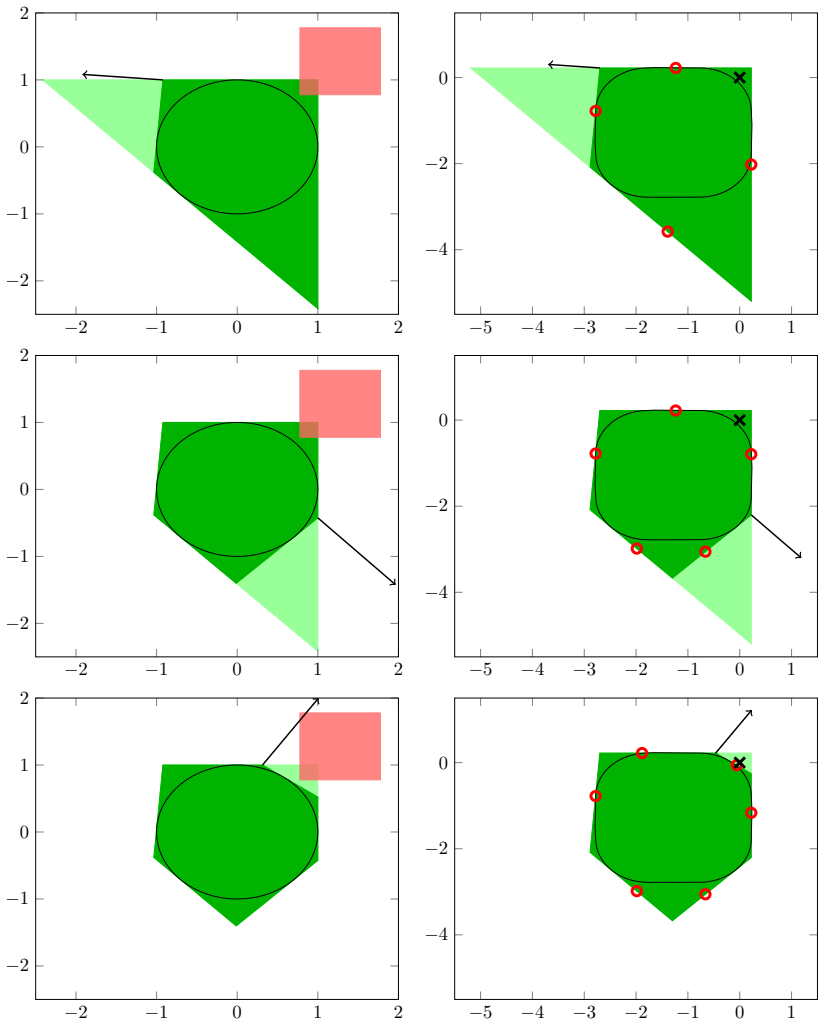


Figure 6.1: Demonstration of the directed approximation algorithm (top to bottom). The left column shows the set $\mathcal{R}$ (black outline), its overapproximation (dark green), and the guard set $\mathcal{S}$ (red box). The right column shows $\mathcal{Q} = \mathcal{R} \oplus (-\mathcal{S})$ (black outline) the outer approximation Q_k (dark green) and the vertices of the inner approximation C_k (red circles). The last iteration shows separation since the origin (black x) lies outside of Q_k

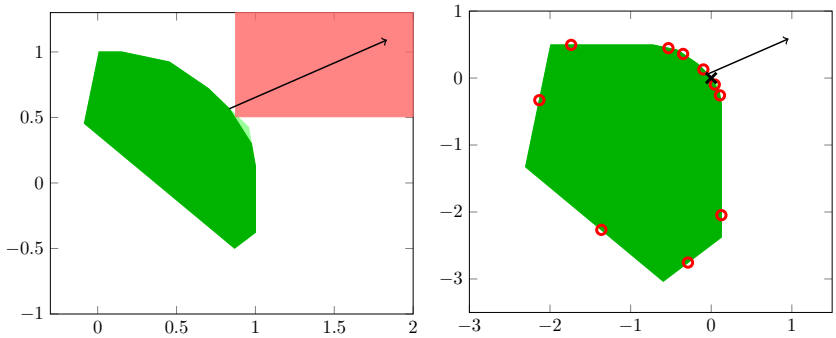


Figure 6.2: Example that shows the directed fashion of the algorithm: facets in the lower left hand corner of $\mathcal{R}$ are no longer refined, since it was shown that refining them will not improve the separation result.

even if $0 \in \mathrm{CH}\{q_0, \ldots, q_k\}$. In this case, we choose the closest point on the *border* of $\mathrm{CH}\{q_0, \ldots, q_k\}$, which tends to "push" facets outwards in a way similar to the Directed Approximation algorithm. Since we need bounded facet slabs, we start with a bounded initial approximation. Given the set $\mathcal{Q}$ and a termination threshold accuracy $\mu_{\min} \geq 0$, our modified GJK algorithm proceeds as follows:

1. Construct an initial, bounded outer approximation to get facet slabs.
   (a) Let $D_{init} = \{d_0, \ldots, d_{m-1}\}$ be a set of directions that span $\mathbb{R}^n$, e.g., the normals of a regular simplex or a bounding box.
   (b) Start from an arbitrary direction d_m. Let $k = m$.
2. *Estimate* a point q_k that maximizes $d_k^\mathsf{T} q$ for $q \in \mathcal{Q}$.
   - Compute $r_k^+ = \mathsf{support}(\mathcal{Q}, d_k, \varepsilon)$. If $r_k^+ < 0$, stop with result "separation".
   - Choose $q_k \in \lfloor \mathcal{Q} \rfloor_k$, e.g., the relative Chebyshev center.
3. Let q_k^* be the point *on the border* of $\mathrm{CH}\{q_m, \ldots, q_k\}$ closest to 0. If $0 \notin \mathrm{CH}\{q_m, \ldots, q_k\}$, let $d_{k+1} = -q_k^*$ (like GJK). Otherwise, let $d_{k+1} = q_k^*$.
4. If $d_{k+1} \in D$, abort since an infinite cycle may take place. Otherwise, add d_{k+1} to D.
   If $|d_k^\mathsf{T} d_{k+1} - \|d_k\| \|d_{k+1}\|| \leq \mu_{\min}$, stop with result "unknown".
5. Let $k \leftarrow k + 1$ and go to step 3.

This modified GJK algorithm may not terminate, or even converge to the point closest to 0. It is presented here because it can detect separation often much faster than Directed Approximation. This will be examined closer in the experimental section.

6.3 Timed Flowpipe Separation

Our goal is to decide whether and when the solutions of the differential equation

$$\dot{x}(t) = Ax(t) + u(t), \qquad u(t) \in \mathcal{U}, \tag{6.9}$$

i.e., the flowpipe $\mathcal{X}_t$, intersect with a given convex set $\mathcal{S}$. We call this the *flowpipe separation problem.* The timed flowpipe separation problem is to identify the time points where the flowpipe is separated from a given (guard) set $\mathcal{S}$. We limit our discussion to a bounded guard set $\mathcal{S}$ and a finite time horizon T. If $\mathcal{S}$ is unbounded, one can render it bounded by computing a coarse flowpipe approximation that is bounded due to finite T, and intersecting $\mathcal{S}$ with this coarse approximation.

Definition 6.4 (Timed flowpipe separation)**.** *Given compact convex sets $\mathcal{X}_0, \mathcal{S} \subset \mathbb{R}^n$ and a time interval $[t_1, t_2]$, a* separating time domain *$\mathcal{T}$ is a subset of $[t_1, t_2]$ such that for all $t \in \mathcal{T}$, $\mathcal{X}_t \cap \mathcal{S} = \emptyset$.*

Knowing the time intervals in which the system enters and leaves the guard can be used to improve the flowpipe approximation. Similarly, timed flowpipe separation can identify at what time t' all trajectories have left the invariant $\mathcal{I}$. Then t' can be taken as time horizon for a more precise flowpipe approximation. The smaller the time domain $\mathcal{T}$, the more precise (and cheaper) the flowpipe approximations can be.

In this section, we present algorithms to decide flowpipe separation with as little computational effort as possible.

6.3.1 Flowpipe Separation using Convexification

Flowpipe separation using convexification is a straightforward application of the convex separation algorithms to a flowpipe approximation consisting of a finite number of convex sets. For each set in the approximation, a convex separation algorithm is executed. If it shows separation on all sets in the sequence, the flowpipe is separated. However, it must be decided when a convex set is accurate enough, or whether it requires being split in several parts.

We now present a separation procedure for a given initial set $\mathcal{X}_0$, a guard set $\mathcal{S}$ and a time interval $[t_1, t_2]$. It uses the convexified flowpipe approximation from Sect. 5.1.3 and a convex separation algorithm from Sect. 6.2.1, and returns a set of convex sets that could not be separated from $\mathcal{S}$.

1. Start with an initial accuracy ε_0 and an initial set of directions $D = \{d_1, \ldots, d_m\}$ that spans $\mathbb{R}^n$, which guarantees that the approximation is bounded.

2. Apply the flowpipe approximation from Sect. 5.1.3 to compute the flowpipe approximation consisting of convex sets $\Omega_0, \Omega_1, \ldots$, using directions D and accuracy ε_0.

3. For each Ω_j, run a convex separation algorithm to separate it from $\mathcal{S}$, where each call to $\mathsf{support}(\Omega_i, d, \varepsilon)$ is implemented as follows:

   (a) Compute upper and lower bounds on the support function of $\mathcal{X}_t$: $(s^+_{d,\varepsilon}(t), s^-_{d,\varepsilon}(t)) = \mathrm{sReach}(\mathcal{X}_0, A, \mathcal{U}, [t_j, t_{j+1}], d, \varepsilon)$.

   (b) Let $\hat{s}(t)$ be the concave hull of $s^+_{d,\varepsilon}(t)$ over the time interval $[t_j, t_{j+1}]$. This corresponds to convexifying the set over this time interval.

   (c) Let $s^+ = \max_{t \in [t_j, t_{j+1}]} \hat{s}(t)$,
   let $\varepsilon_{\text{result}} = \max_{t \in [t_j, t_{j+1}]} \hat{s}(t) - s^-_{d,\varepsilon}(t)$.

   (d) If $\varepsilon_{\text{result}} \leq \varepsilon$, use s^+ as support value for the support function of Ω_j.

   (e) Otherwise, a single set does not suffice to represent the flowpipe with sufficient accuracy. Divide $[t_j, t_{j+1}]$ into subintervals such that on each interval, the concave hull of $s^+_{d,\varepsilon}(t)$ satisfies the conditions of step 3c and 3d. Replace Ω_j by restrictions of Ω_j to those subintervals. For each, apply the convex separation algorithm again.

4. Return the Ω_j, for which separation could not be shown.

6.3.2 Flowpipe Separation Point-Wise over Time

A convex separation algorithm can solve the flowpipe separation problem for any given point in time t^*, since we know that $\mathcal{X}_{t^*}$ is convex. However, we need to extend separation to intervals of time. With Lemma 6.2, the following criterion is straightforward.

Lemma 6.5. *The flowpipe $\mathcal{X}_{t_1,t_2}$ is separated from a convex set $\mathcal{S}$ if and only if for all $t \in [t_1, t_2]$ there exists a direction $d_t \in \mathbb{R}^n$ such that*

$$\rho_{\mathcal{X}_t}(d_t) + \rho_{\mathcal{S}}(-d_t) < 0. \tag{6.10}$$

The question is therefore how to find a suitable direction d_t for each point in time. We present two ways for applying separation over an interval of time: first, keeping the direction d fixed over time, and second, letting d_t evolve over time according to the dynamics of the system.

Separating with Fixed Direction

We use the bounds on the support function of $\mathcal{X}_t$ as described in Sect. 5.1.3. Given a time interval $[t_b, t_e]$ and a fixed direction d, let

$$\mathrm{sep}_{\mathcal{S},\mathcal{X}_0,A,\mathcal{U},d}(t) = \rho_{\mathcal{X}_t}(d) + \rho_{\mathcal{S}}(-d). \tag{6.11}$$

Applying (6.10), the flowpipe is separated from $\mathcal{S}$ for any $t \in [t_b, t_e]$ for which

$$\mathrm{sep}_{\mathcal{S},\mathcal{X}_0,A,\mathcal{U},d}(t) < 0. \tag{6.12}$$

Using the approach from Chapter 5 we can compute a bound $^{+}_{d,\varepsilon}(t)$ on the support function of $\mathcal{X}_t$ for a given precision $\varepsilon > 0$, i.e.,

$$s^+_{d,\varepsilon}(t) - \varepsilon \leq \rho_{\mathcal{X}_t}(d) \leq s^+_{d,\varepsilon}(t).$$

Let r^+_d be an upper bound $r^+_d \geq \rho_{\mathcal{S}}(-d)$. Since

$$\mathrm{sep}_{\mathcal{S},\mathcal{X}_0,A,\mathcal{U},d}(t) < s^+_{d,\varepsilon}(t) + r^+_d,$$

the function $s^+_{d,\varepsilon}(t)$ is a witness that $\mathcal{X}_t$ is separated from $\mathcal{S}$ at any time point $t \in [t_b, t_e]$ at which

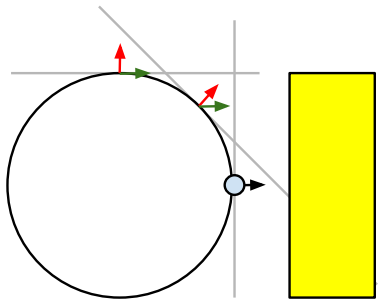
$$s^+_{d,\varepsilon}(t) + r^+_d < 0.$$

The following example shall illustrate that there are cases where *only* a single, fixed, direction (or an arbitrarily small neighborhood around it) can show separation.

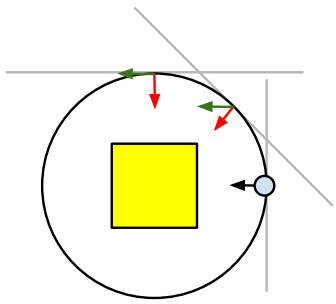
Example 6.6. *Consider the example shown in Fig. 6.3(a). The initial set $\mathcal{X}_0$ consists of a single point, and the flowpipe consists of the point moving around the origin in a circle. The direction $d = (1, 0)$ shows separation even over an unbounded time horizon, as indicated by the green arrows. By making $\mathcal{S}$ large enough in the vertical direction, we can reduce the set of separating directions to an arbitrarily small neighborhood of d. A guard may not be separable by a single fixed direction over the entire time horizon, as shown in Fig. 6.3(b).*

We can characterize precisely when separation with a fixed direction is useful:

Lemma 6.7. *A fixed direction $d \in \mathbb{R}^n$ separates a flowpipe $\mathcal{X}_{t_b,t_e}$ from a convex set $\mathcal{S}$ if and only if $\mathrm{CH}(\mathcal{X}_{t_b,t_e})$ and $\mathcal{S}$ are separated by d.*



(a) Keeping the direction that separates the initial set shows separation over the entire time horizon



(b) Dynamically adapting the direction that separates the initial set shows separation over the entire time horizon

Figure 6.3: In both examples, separation can be shown for the entire flowpipe after finding a separating direction for the initial set

Separating with Dynamic Direction

Instead of keeping the direction fixed over time, we can let it evolve according to the dynamics of the system. Given a direction d_0, let $d_t = e^{-A^{\mathsf{T}}t} d_0$. The dynamic direction d_t evolves with the system in the following sense. If d_0 is a normal vector of $\mathcal{X}_0$ at a point $x^*(0)$ (meaning it is tangent to $\mathcal{X}_0$ and touches at x_0^*), then d_t is a normal vector of $\mathcal{X}_t$ at a point $x^*(t)$. E.g., if $\mathcal{X}_0$ is a polytope and d_0 is a facet normal of $\mathcal{X}_0$, then d_t is a facet normal of $\mathcal{X}_t$. The following example shall illustrate where a single dynamic direction can show separation over the entire time horizon.

Example 6.8. *Consider the example shown in Fig. 6.3(a). Starting from a direction that separates the initial set, the dynamic direction (red arrows) shows separation for only a small amount of time, which is even smaller if the guard set is larger in the vertical direction. For the guard set shown in Fig. 6.3(b), the dynamic direction separates over the entire time horizon, while no fixed direction can do so.*

We can reduce the separation along a dynamic direction to the separation of a fixed direction, which allows us to apply the techniques from the previous section.

Lemma 6.9. *$\mathcal{X}_t$ is separated from $\mathcal{S}$ in direction $d_t = d_0^{\mathsf{T}} e^{-At}$ if and only if $\mathrm{sep}_{-\mathcal{S},-\mathcal{X}_0,-A,\mathcal{U},d_0}(t) < 0$ or, equivalently, $\mathrm{sep}_{\mathcal{S},\mathcal{X}_0,-A,-\mathcal{U},-d_0}(t) < 0$.*

For clarity, we use the following notation. Let $\mathrm{Reach}_t(\mathcal{S}, A, \mathcal{U})$ be the (forward) reachable set from $\mathcal{S}$ under the dynamics (6.9). The *backwards reachable set* can be computed as a forward reachable set by transforming the dynamics, since

$$\mathrm{Reach}_{-t}(\mathcal{S}, A, \mathcal{U}) = \mathrm{Reach}_t(\mathcal{S}, -A, -\mathcal{U})$$

is the set reachable from $\mathcal{S}$ going backwards in time. Applying this interpretation to Lemma 6.9, separating $\mathcal{X}_0$ from $\mathcal{S}$ by forward reachability with a dynamic direction is equivalent to separating $\mathcal{S}$ from $\mathcal{X}_0$ by backward reachability with a fixed direction.

As a corollary of Lemma 6.7, we can characterize when a dynamic direction is useful:

Corollary 6.10. *A flowpipe $\mathcal{X}_{t_b,t_e}$ can be separated from $\mathcal{S}$ with a dynamic direction $d_t = d_0^{\mathsf{T}} e^{-At}$ if and only if the convex hull of the backwards reachable set from $\mathcal{S}$ is separated from $\mathcal{X}_0$ in direction $-d_0$.*

Pointwise Separation Algorithm

We now describe an algorithm that uses a convex separation algorithm pointwise in time to separate the flowpipe from a guard set $\mathcal{S}$ over a time interval $[t_1, t_2]$.

The algorithm takes as input the system description, the initial set $\mathcal{X}_0$, the guard set $\mathcal{S}$, a time interval $[t_1, t_2]$. It returns a set of time intervals for which separation could not be shown.

1. Picking some $t^* \in [t_1, t_2]$, e.g., the midpoint, we use a convex separation algorithm on $\mathcal{X}_{t^*}$ to detect or refute separation at time t^*. If separation cannot be shown, stop. Otherwise, we obtain a separating direction d and a bound ε on the required accuracy.

2. Compute $s^+_{d,\varepsilon}(t) = \text{sReach}(\mathcal{X}_0, A, \mathcal{U}, [t_1, t_2], d, \varepsilon)$ and $p^+_{d,\varepsilon}(t) = \text{sReach}(-\mathcal{S}, -A, \mathcal{U}, [t_1, t_2], d, \varepsilon)$.
3. Remove from $[t_1, t_2]$ the t where $s^+_{d,\varepsilon}(t) + \rho_{\mathcal{S}}(-d) < 0$ or $p^+_{d,\varepsilon}(t) + \rho_{-\mathcal{X}_0}(-d) < 0$ (separation holds).
4. For each of the remaining sub-intervals, apply the pointwise separation algorithm recursively and return the obtained intervals.

The algorithm has the weakness that it may stop prematurely if the separation time t^* is poorly chosen. We propose two improvements: First, the algorithm may be repeated on the subintervals $[t_1, t^*]$ and $[t^*, t_2]$, until their size falls below a given threshold. Second, the algorithm may be applied a second (and third) time, choosing t^* to be the start (and end) times of the intervals instead.

Indeed, separating on start and end times may reduce the size of the flowpipe segments, for which discrete successor states are computed, and thus improve the approximation accuracy of the reachability algorithm even in cases where separation could not be shown.

6.4 Experimental Results

In this section, we evaluate the presented algorithms on two classes of benchmarks. We have implemented the algorithms in the SpaceEx hybrid model checker. We conducted the experiments on a machine with an Intel i7 3.4 GHz processor and 16 GB of RAM. We illustrate the results for the convex separation algorithms from Section 6.2 on the *sphere* benchmark. Here, we consider an overapproximation of an n-dimensional sphere by a polytope with m constraints. The guard set is a single point. Recall that with Lemma 6.2, any convex separation problem can be reduced to separating one convex set from a single point (the origin). Our benchmark is thus equivalent to separating two ball-approximations (each with half the radius), which we consider to be a challenging instance.

We consider a number of benchmark instances by varying the dimension of the sphere n, the number of constraints m as well as the distance to the guard. By varying those parameters we can flexibly adjust the benchmark instance complexity. The guard is defined based on the distance and the normal of the guard hyper-plane. For every tuple $(n, m, \text{distance})$ we pick 10 random vectors to be used as guard normals. We analyze every instance using the adapted GJK algorithm and the directed approximation algorithm. Note that we accumulate the results over those 10 random vectors for every tuple $(n, m, \text{distance})$. In particular, we report the relation of the number of the benchmark instances where a convex separation algorithm has found the right answer before the time-out (success rate), the minimum, maximum and average numbers of direction refinements and run-time, respectively (see Table 6.1). We terminate the analysis when either the timeout of 500 s or the maximum number of 500 refinement iterations has been reached.

We observe that the time needed to show the separation generally increases with the sphere dimensionality and the number of constraints. Furthermore, the number of direction refinements to show separation increases as well. We note that the run-time complexity also depends on the chosen guard. This can

be seen in the column "✓ %" where the success rate is reported. For example, for the instance 4, we succeed in 55% of the cases. Moreover, the distance to the guard plays an important role: The smaller the distance, the more complex it becomes to find a separating plane. The success rate goes from 5% for instance 24 with the distance equal to 0.1 up to 100% for distance 1.

The GJK algorithm proves separation in most cases in a short time. The DA algorithm performed very well for the smaller dimensions and number of constraints, still scales worse than the GJK algorithm. We note that the DA algorithm provides more guarantees compared to the GJK algorithm. Furthermore, the DA algorithm can provide sound results for the cases with overlap.

We examine the flowpipe separation algorithms from Section 6.3 based on the circle benchmark. In this setting, we iteratively call the convex separation algorithm for every time interval where the system is expected to enter and leave the guard. In the circle benchmark, we model an object which moves on a circle orbit in 2D space, i.e, its dynamics are provided by the following differential equations: $\dot{x} = -y \wedge \dot{y} = x$. The system behavior is illustrated in Figure 6.3. We take a segment between two points on the orbit as an initial region. We consider two positions of a rectangular guard: the guard is inside the circular orbit (GI; see Figure 6.3(b)) and the guard is outside (GO; see Figure 6.3(a)). The results for the circle benchmark are presented in Table 6.2.

We observe that with the GO-instances (guard outside), we need to consider much less time intervals. Furthermore, the number of directions to prove the separation drastically differs. For example, the directed approximation algorithm with flowpipe separation using convexification needs 5 direction refinements if the guard is outside. However, the number of refinements increases to 108 if the guard is located inside. Note that the algorithm also cannot prove the separation in this case.

For the GO-instances (guard inside), there exists at least one separating plane for *all* the time intervals. However, if the guard is located inside, we observe that *no* plane exists which separates the guard from the flow-pipe for all time intervals. Therefore, the algorithm needs to search for an *individual* separating plane for every time interval for the GI-instances. Therefore, we expect the flowpipe separation using convexification to be particularly useful for GO-instances. This hypothesis is confirmed by our experiments where all the convex separating algorithms require only one interval refinement. However, due to the same reasons, the flowpipe separation performs badly on the GI-instances. Moreover, the convex separating algorithms cannot even find the separating plane because the convexification leads to rather over-approximative results.

Table 6.1: Experimental results for the sphere benchmark

						# Direction Ref.			Runtime		
ID	Alg.	n	m	Dist.	✓ %	min.	max.	avg.	min.	max.	avg.
1	GJK	2	16	0.01	100%	2	52	13.550	0s	0.077s	0.015s
2	GJK	3	36	0.01	95%	7	54	30.789	0.008s	0.125s	0.061s
3	GJK	4	64	0.01	85%	20	110	67.588	0.056s	0.895s	0.446s
4	GJK	5	100	0.01	55%	105	162	133.909	1.402s	6.689s	3.027s
5	GJK	2	16	0.1	95%	1	19	7.157	0s	0.021s	0.005s
6	GJK	3	36	0.1	100%	2	34	12.000	0s	0.067s	0.017s
7	GJK	4	64	0.1	95%	2	56	18.263	0.001s	0.241s	0.057s
8	GJK	5	100	0.1	65%	9	40	18.833	0.027s	0.288s	0.104s
9	GJK	2	16	0.5	100%	1	3	1.900	0s	0.002s	0s
10	GJK	3	36	0.5	100%	1	11	3.800	0s	0.015s	0.003s
11	GJK	4	64	0.5	100%	1	21	6.300	0s	0.057s	0.013s
12	GJK	5	100	0.5	95%	2	27	7.000	0.001s	1.300s	0.089s
13	GJK	2	16	1	100%	1	2	1.650	0s	0.001s	0s
14	GJK	3	36	1	100%	1	3	2.150	0s	0.002s	0s
15	GJK	4	64	1	100%	1	8	2.850	0s	0.015s	0.002s
16	GJK	5	100	1	100%	2	8	3.150	0.001s	0.026s	0.004s
17	DA	2	16	0.01	100%	2	10	6.500	0.003s	0.047s	0.023s
18	DA	3	36	0.01	95%	30	116	46.368	0.707s	33.642s	3.553s
19	DA	4	64	0.01	40%	150	210	180.875	142.302s	470.624s	309.943s
20	DA	5	100	0.01	0%	500	500	—	—	0s	—
21	DA	2	16	0.1	100%	2	7	3.750	0s	0.027s	0.008s
22	DA	3	36	0.1	100%	7	40	18.800	0.030s	1.421s	0.306s
23	DA	4	64	0.1	95%	13	161	88.210	0.207s	188.231s	51.952s
24	DA	5	100	0.1	5%	116	116	116.000	188.580s	188.580s	188.580s
25	DA	2	16	0.5	100%	2	4	2.450	0s	0.010s	0.002s
26	DA	3	36	0.5	100%	2	13	5.500	0s	0.103s	0.028s
27	DA	4	64	0.5	100%	2	83	22.900	0.001s	23.274s	2.704s
28	DA	5	100	0.5	85%	2	113	28.000	0.001s	179.145s	20.968s
29	DA	2	16	1	100%	2	2	2.000	0s	0.002s	0s
30	DA	3	36	1	100%	2	10	3.150	0s	0.079s	0.009s
31	DA	4	64	1	100%	2	31	7.300	0s	1.396s	0.182s
32	DA	5	100	1	100%	2	40	9.050	0s	7.546s	0.671s

ID: benchmark instance ID, Alg.: convex separation algorithm, n: dimension of the sphere, m: number of facets in the over-approximation of the sphere, Dist.: distance between the guard and the polytope which over-approximates the sphere, ✓ %: the relation of the number of the benchmark instances where a convex separation algorithm has found the right answer (until OOT = 500 s.) to the total number of benchmarks in this class, # Direction Ref.: number of directions the separation algorithm has tried out (minimum, maximum and average values), Runtime: runtime in seconds (minimum, maximum and average values), DA denote the directed approximation algorithm, GJK stands for the adapted GJK algorithm.

Table 6.2: Experimental results for the circle benchmark.

flowpipe sep.	convex sep.	# Calls	# d	Runtime	Result
guard outside the circular orbit					
CH	GJK	1	7	0.038s	SEP
CH	DA	1	5	0.046s	SEP
PW	GJK	1	6	0.042s	SEP
PW	DA	1	6	0.060s	SEP
guard inside the circular orbit					
CH	GJK	9	1767	37.903s	INT
CH	DA	14	108	0.749s	INT
PW	GJK	19	258	0.960s	SEP
PW	DA	18	128	0.791s	SEP

CH: flowpipe separation using convexification. PW: pointwise flowpipe separation. GJK: adapted GJK algorithm. DA: directed approximation algorithm. SEP: separation shown. INT: intersection shown.

Chapter 7

The SpaceEx Verification Platform

The verification of hybrid systems requires ingredients taken from the classical verification of transition systems, augmented with new special techniques for doing verification-like operations (successor computation) on the continuous dynamics. Early tools [55, 6] focused on relatively simple continuous dynamics in each discrete state, where the derivative of the continuous variables does not depend on their values. For such "linear" hybrid automata, the computation of successors in the continuous domain can be realized by linear algebra. Nevertheless, it turned out that switching between such simple continuous modes one can easily construct undecidability gadgets and hence the *exact* verification of hybrid systems turned out to be a dead end.

The second wave of hybrid verification tools [26, 10, 60] indeed abandoned exact computations and focused more on computing *approximations* of the reachable states for systems admitting less trivial continuous dynamics. Such techniques and tools could handle hybrid systems with continuous dynamics defined by linear differential equations with inputs. However, the size of systems that could be handled was modest, typically very few continuous state variables. Another thread in hybrid verification focuses on systems with a very large discrete state-space with very few continuous variables. Typically such models are aimed to verify the code of computerized control systems, with a very modest modeling of the external environment. The major preoccupation in these efforts is in combining the continuous part with techniques, such as BDD or SAT, for handling large discrete state-spaces [29, 80].

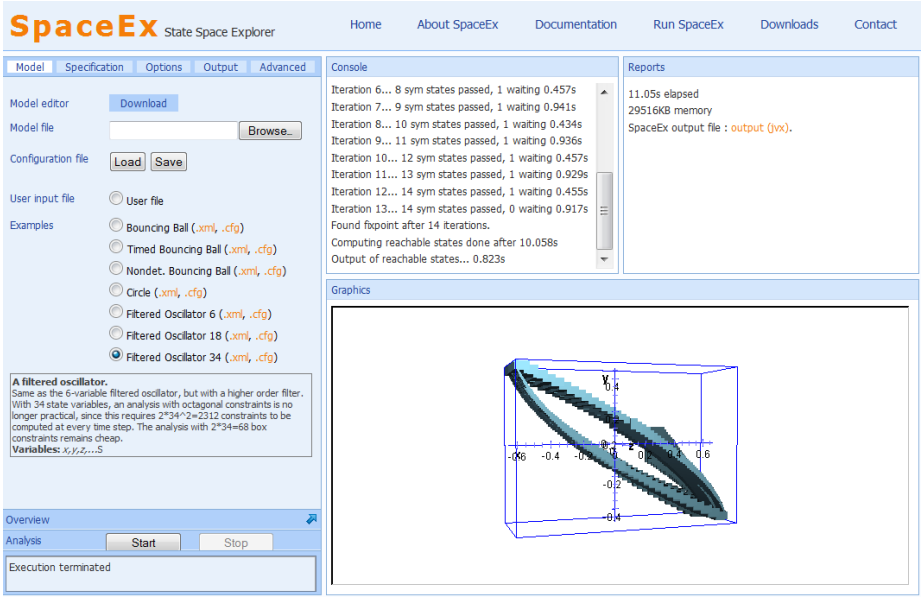
Algorithmic improvements and the use of *support functions* [66, 65, 67] have dramatically increased the scope of linear systems that can be verified to several hundreds of state variables. Moreover, significant advances have been made on applying these linear techniques to nonlinear systems via "hybridization" (approximating a nonlinear system by a piecewise-affine one) [8], which have led to the verification of non-trivial nonlinear systems [31].

In order to transfer these research achievements into a robust and user-friendly toolset, we developed SpaceEx, an extensible verification platform for hybrid systems. It features implementations of many of the above-mentioned developments, a graphical model editor and a graphical user interface. This

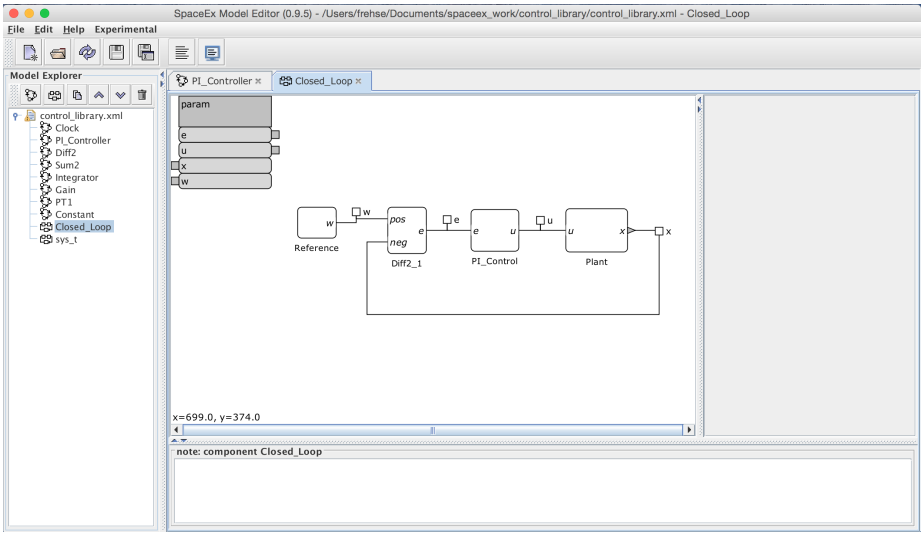
transfer is not only about software engineering, modularity and user interfaces: it consists in improving and fine-tuning the algorithms to make them applicable to real-world problems. SpaceEx consists of three components:

- The *analysis core* is a command line program that takes a model file, a configuration file that specifies the initial states, the scenario and other options, and then analyzes the system and produces a series of output files.
- The *web interface*, shown in Fig. 7.1(a), is a graphical user interface with which one can comfortably specify initial states and other analysis parameters, run the analysis core, and visualize the output graphically. The web interface is browser-based, and accesses the analysis core via a web server, which may be running remotely or locally on a virtual machine.
- The *model editor*, shown in Fig. 7.1(b), is a graphical editor for creating models of complex hybrid systems out of nested components.

The SpaceEx platform is publicly available at `spaceex.imag.fr`. In this chapter, we describe the architecture of the SpaceEx platform, its input language, and its model editor. Further information about modeling and verifying systems in SpaceEx is given in [36].



(a) Web interface



(b) Model editor

Figure 7.1: Graphical user interfaces of the SpaceEx platform

7.1 SpaceEx Architecture

The SpaceEx architecture was designed by comparing a variety of existing verification tools and algorithms, which fit the set-based reachability algorithm described in Sect. 2.2. We first report on the comparison and then present the chosen architecture and its typical execution.

7.1.1 Comparison of Tool Architectures

In this section, we give an overview of the architectural tool comparison in [45]. Note that while the analysis was carried out in 2008, architectures do not seem to have changed since. A majority of recent tools like CORA [1], Flow* [24], and HyCreate [14] fall into the same category and fit the SpaceEx architecture. The following approaches were considered in our analysis:

- Constant continuous and affine discrete dynamics
  - A HyTech [55]
  - B PHAVer [35]
- Affine continuous and discrete dynamics
  - C d/dt [9]
  - D Using zonotopes [49]
  - E Using support functions [50]
  - F Algorithmic improvements to compute post_c for D,E [51]
- Nonlinear dynamics and abstraction refinement
  - G Approximating nonlinear dynamics by hybridization [8]
  - H Forward/backward refinement [43]
  - I CEGAR-type approaches [41]

The reachability techniques in A,B are for piecewise constant derivatives, exact as well as overapproximative over an infinite time horizon. In C, affine continuous and discrete dynamics are overapproximated by discretizing time over a finite time horizon. In D and E, this technique is improved by exploiting the advantages of a particular representation of continuous sets, plus some low-level algorithmic improvements. In G, the techniques for affine dynamics are extended to nonlinear dynamics by overapproximation based on partitioning the state space. In H, a very simple abstraction/refinement technique is used for deciding safety, and more sophisticated ones based on *counter example guided abstraction refinement* (CEGAR) can be found in I. Approaches A–E constitute low-level algorithms that deal with computing post-images for particular dynamics, while G–I are high-level techniques that use low-level reachability algorithms as an intermediate step.

An analysis of common elements and differences shall provide us with the basis for our design. The system under examination is described as a network of interacting automata. The specification consists of the set of initial and (for safety) forbidden states. In addition, the user has to provide analysis parameters

such as discretization time steps or partition sizes. For the analysis, a parallel composition operator transforms the automaton network into a single automaton, possibly on the fly. The set of reachable states is computed using some variant of the symbolic state algorithm in Sect. 2.2. The resulting set of states undergoes some basic processing (intersection with forbidden states, projection onto variables of interest), and is output to a file or visualized.

The approaches we consider differ along the following lines:

Set Representations Polyhedra (A,B), zonotopes (D), and support functions (E) have each various advantages and disadvantages on fundamental set operations. For example, for polyhedra in constraint form computing intersection is cheap and Minkowski sum is expensive, while for zonotopes Minkowski sum is cheap and the intersection of two zonotopes is not generally a zonotope.

Discrete post-computations Various exact as well as overapproximative techniques for computing the image of discrete transitions are used and depend on the set representation. Most techniques apply to discrete dynamics in the form of affine maps (resets). Most approaches use some type of clustering, such as taking the convex hull of all sets that intersect with the guard set.

Continuous post-computations The states reachable by time elapse are generally overapproximated. Different techniques are applicable according to the type of continuous dynamics as well as the set representation. For A,B the image is over infinite time, while C,D,E,F discretize time and compute it over a bounded interval. Even for just linear dynamics, variations abound. For example, F avoids the wrapping effect by essentially reordering the computation and its approach is applicable to D,E.

State exploration Most approaches are defined for forward reachability, but can equally be applied as backward reachability by reversing the system dynamics. One direction may work better than another depending on the characteristics of the system [76], and H combines both. I requires keeping track of the dependency graph between symbolic states, i.e., which are the successor states of which. The explored states need to be stored in some form of passed/waiting list, and at each iteration the explored states need to be separated into those that are new and those that already been explored, which involves some form of difference operation (exact, overapproximative, see A).

Model transformations Hybridization (G) and abstraction/refinement techniques (B,H,I) involve duplicating (splitting) locations, adding and removing transitions, and modifying dynamics and invariants. Such changes in the model must be compatible with the state exploration if they are to be carried out on the fly, or if state representations are to be compatible with different variants of the same model.

High level algorithms In abstraction/refinement schemes like H or I, computing the reachable states is just one step in a larger process. They require certain low-level information like the dependency graph and counter examples to be accessible, and they entail model transformations.

Automaton composition Composition operators differ in the type of communication (synchronization) and how variables are shared (A versus B). The

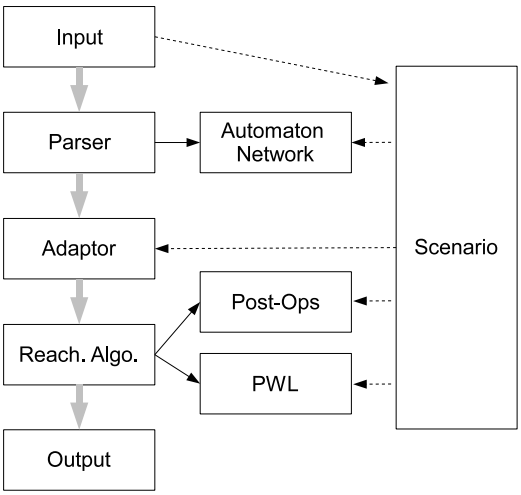


Figure 7.2: Schematic of the tool architecture (solid arrows represent acquaintance between objects, dashed arrows represent instantiation). Grey arrows indicate in which order the different components are executed

composition should take place on the fly and on demand in order to avoid the construction of the full product automaton, which is typically too large to be represented explicitly.

We designed an architecture capable to host the above approaches, exploiting the common elements and leaving room for the differences. This architecture is presented in the next section.

7.1.2 Architecture and Execution

Based on the survey in the previous section, we define the following key elements and operations for a reachability tool:

- automaton representation: add locations, transitions
- automaton network representation (controls composition; itself an automaton): add automata
- discrete and continuous set representations: inclusion and emptiness tests, transforms (intersection, affine maps, etc.)
- adapt: convert sets and dynamics to the right form (if possible)
- Passed-and-waiting-list (PWL): add, pop symbolic states
- continuous-post: transform a symbolic state into a set of symbolic states
- discrete-post: transform a symbolic state into a finite set of symbolic states

The implementation choices depend on each other. E.g., a specific continuous-post operator might only apply to affine dynamics and require polyhedra as set

representations. At the same time, we would like to keep the concrete classes encapsulated as much as possible; whoever writes the polyhedron class shouldn't need to know anything about hybrid automata. In the SpaceEx architecture, each of the above elements is represented by an abstract base class, from which implementations must be derived. We call a set of implementations for these elements a *scenario*, and use a scenario object as a factory for them, similar to the strategy design pattern in [47]. A developer can provide his or her own implementation for each element, and define the global set-up via a scenario object. A run of the tool consists of the following steps, as shown in Fig. 7.2:

1. The user provides the input : models (XML), user commands, scenario selection, output selection.
2. The input file is parsed to generate a general representation of the automata (transitions/locations) and sets (initial states, bad states).
3. The general automata are adapted to the right set representation and dynamics according to the scenario (adapt).
4. The automaton network is instantiated according to the scenario.
5. The user selected algorithm (reachability, safety) is executed, using the elements provided by the scenario (PWL, post).
6. The output is created : visualization, file export (model, states).

User options are used to select the scenario, additional options can be passed directly to the scenario.

7.2 Scenario Implementations

A scenario provides implementations for the all key elements listed in Sect. 7.1.2. The currently available scenarios are as follows:

- PHAVer: This scenario implements the basic reachability algorithms of the tool PHAVer [35]. The dynamics are piecewise constant, sets are polyhedra implemented using infinite precision arithmetic by the Parma Polyhedra Library [12], and successor operations are implemented by geometric operations on polyhedra in dual description. The resulting reachable set is the exact solution.
- LGG: This scenario implements the support function approach from Chapter 3. The dynamics are piecewise affine. Sets are polyhedra in constraint representation, implemented using SpaceEx data structures in floating point representation. An additional data structure represents sets of template polyhedra in a compact form. Set representations are approximate, with user-specified tolerances on the absolute and relative floating point errors. The resulting reachable set is an overapproximation modulo floating point errors.
- STC: This scenario implements the space-time approach from Chapter 5. The dynamics are piecewise affine. Sets are polyhedra in constraint representation, taken from the LGG scenario, custom data structures for

efficiently representing flowpipe samples via piecewise linear scalar functions, and generic data structures for representing support functions. The resulting reachable set is an overapproximation modulo floating point errors.

- Simulator: This scenario implements a rudimentary numerical simulation algorithm, cast into the reachability framework. The dynamics are nonlinear. Sets are sets of points in the state space. The time successor operations are handled by an ODE solver, here CVODE [58]. Containment checks and clustering are handled by treating each state as representative for a neighborhood of given diameter.

These scenarios share implementations for discrete sets, automata, automata networks and the PWL. Each provides its own version of continuous sets, dynamics, adaptors, and post-operators.

7.3 Modeling in SpaceEx

We provide a brief overview of how hybrid systems are modeled in SpaceEx and its Model Editor. For details, see [36, 28]. SpaceEx models are stored in the `sx` format, an XML based format for which there is a graphical model editor. `sx` models are similar to the hybrid automata known in literature, except that they provide a richer mechanism of hierarchy, templates and instantiations. An `sx` model consists of one or more components. When SpaceEx reads an `sx` file, it translates the components into either a hybrid automaton or into a network of hybrid automata in parallel composition. So, for the purposes of analysis, a component defines a hybrid automaton, with everything else (hierarchy etc.) being syntactic sugar whose only purpose is making the construction of complex models easier.

7.3.1 Base and Network Components

A model is made up of one or several *components*. There are two types of components: A *base component* corresponds to a single hybrid automaton as defined in Sect. 2.1.2. A *network component* consists of one or more instantiations of other components (base or network) and corresponds to a set of hybrid automata in parallel composition. Every component has a set of *formal parameters*. A formal parameter may be a continuous variable or a synchronization label. A formal parameter is part of the *interface* of a component, unless it is declared as *local* to the component.

Example 7.1. *The interface of a bouncing ball model is shown in Fig. 7.3. The continuous state variables are x and v. The constants g, c, and eps are declared as variables with constant dynamics. They will be assigned values when the template is instantiated inside a network component. The last parameter is the synchronization label hop.*

Instantiating Components When a component A is instantiated inside a network component B, a copy of A is created and added to the contents of

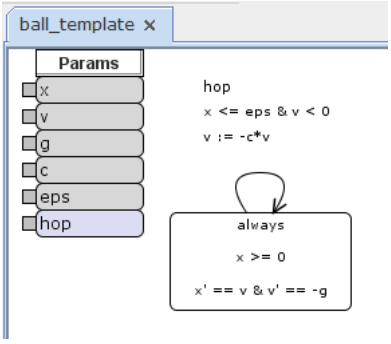


Figure 7.3: A bouncing ball model with its formal parameters: state variables x and v, constants g, c, and eps, and label *hop*

network B. The copy must have a name that is unique within B, say A1. It is a copy by reference, so that any later modifications to A will also apply to A1. Component A can be instantiated several times inside B, say as A1, A2, A3, etc. When instantiating a component A in network B, we must specify what happens to each of the formal parameters in its interface. This is called a *bind*. Every formal parameter of A must be bound to either a formal parameter of B or to a numeric value.

Example 7.2. *Figure 7.4 shows the network component* system*, which is made up of an instantiation of the base component* ball_template*. The instantiation is called* ball*. The formal parameters x, v, and hop are bound to formal parameters x, v, and hop of* system*. The constants g, c, and eps are bound to numeric values.*

Connecting Components Components inside a network can be connected by binding their variables or labels to the same symbols in B. Binding two or more variables to the same variable in a network component means that they become literally the same variable. Their dynamics can be defined in one or several of the components as long as this creates no contradictions.

Example 7.3. *Figure 7.5(a) shows a hybrid automaton model of a PI-controller. It has e as input, e_int as internal variable and u as output. The controller parameters are K_P and K_I., c is a value determining the cut-off frequency. The network component Closed_Loop shown in Fig. 7.5(b) instantiates this controller in a closed-loop system. The PI-controller is connected to other components via shared variables e and u.*

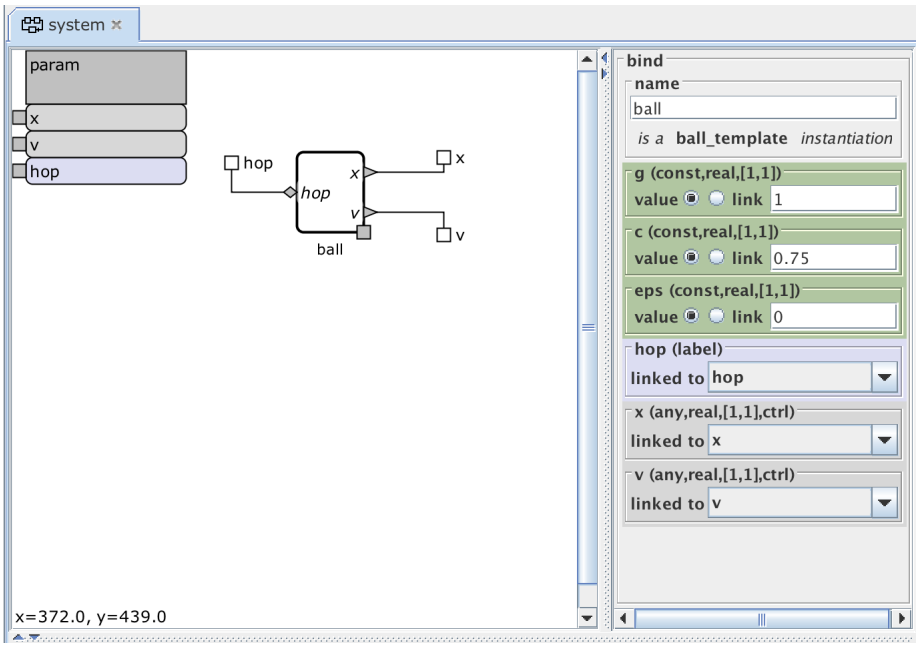
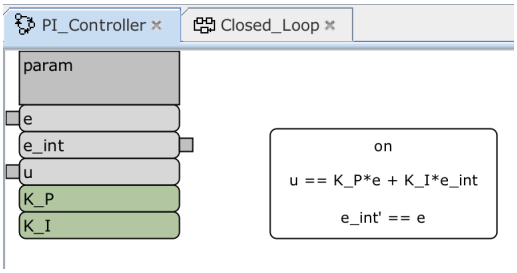
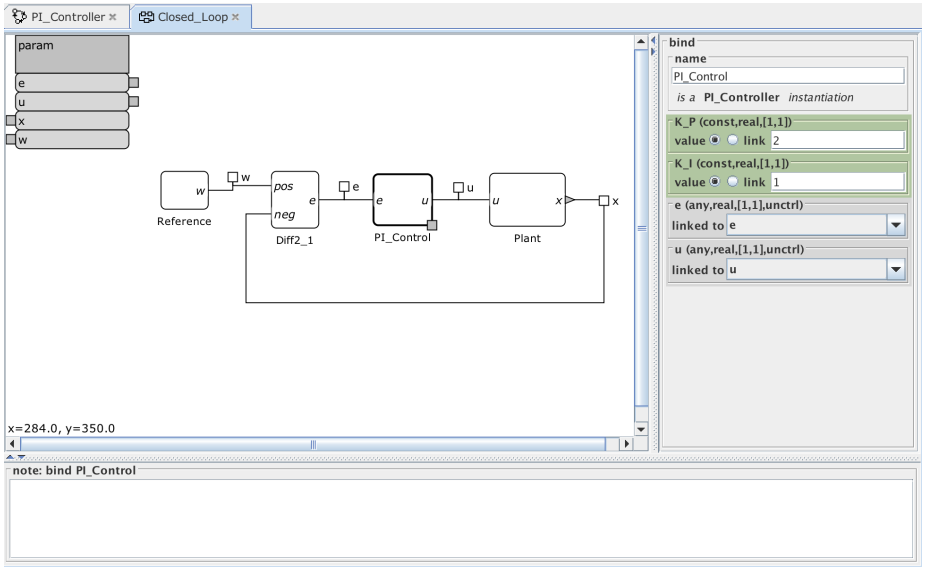


Figure 7.4: The bouncing ball template instantiated in the network component *system*. The constants are bound to the desired values, the other formal parameters are passed on to the interface of system.



(a) hybrid automaton model of a PI controller



(b) closed-loop system

Figure 7.5: An SpaceEx model of a standard feed-back control system

7.3.2 Causality and Shared Variables

SpaceEx models have nondeterministic, acausal semantics: Any component can declare any variable as one its formal parameters, and impose constraints (including equalities) on the variable and its derivative. Different components can impose constraints on the same variable, at the same time or in alternation. In certain application areas, like mechanics or electric circuits, this allows one to decompose models into reusable building blocks, or build models directly from first principles. For example, one component could impose $\dot{x} \leq 0$ and another $\dot{x} \geq 0$. The resulting behavior has to satisfy both, so $\dot{x} = 0$. If the constraints contradict each other, resulting in, e.g., $\dot{x} \in \emptyset$, there is no solution to the differential equations (or inclusions) and thus there might not be any trajectories after a certain point in time. We say that "time stops" in the model. This may or may not be a modeling error.

Because of the acausal semantics, there are no inputs and outputs in SpaceEx models per se. For compositional reasoning, there is the notion of a *controlled variable*, which is closely related, and SpaceEx enforces compositional semantics for this purpose. As a rule of thumb, state variables, typically those whose derivatives are explicitly defined, should be controlled while algebraic variables should be uncontrolled. For more details, see [33].

7.3.3 Initial and Forbidden States

The specification of a reachability problem includes the set of initial states, from which all behavior of the system originates. We consider the initial states to be problem- rather than model-specific, so they are specified in a configuration file rather than the model file. A set of states can be specified in SpaceEx as a boolean combination of *location constraints* and linear constraints over the variables. A location constraint denotes that a given component is or is not in a specific location. The specification of a set of forbidden states is optional. If the reachability analysis terminates, then the output is restricted to the intersection of the forbidden with the reachable states. In addition, the textual output of the tool reports whether the forbidden states are reachable or not.

Chapter 8

Conclusions and Outlook

Set-based reachability analysis can provide valuable insight into the critical behaviors of a system. As a verification technology, it can be used to show the absence of critical behavior, or detect and identify critical paths in the system. However, the verification problem is undecidable except for the most simple hybrid systems, and overapproximations are hard to avoid in algorithmic approaches. The potential uses of set-based reachability go beyond the black-and-white world of traditional verification. Seen as an extension of numerical simulation to sets, it allows one to execute the system symbolically, tracing the evolution of the system not one state at a time, but for groups of behaviors. One can think of this a debugging, and in particular in the context of cyber-physical systems, such functionality may come natural to software developers. It should not be overlooked that set-based reachability can also provide quantitative information about the system. Conservative bounds on critical variables such as temperatures, as well as performance criteria such as rise times, can assist the design process. This may be particularly relevant for component reuse, where thanks to the capacity to include nondeterminism, such bounds can be propagated along a chain of connected components, or passed on through a hierarchy of models. Ensuring the consistency between those models remains a challenge, but reachability may offer one viable technique here, too.

The main challenge for reachability of hybrid systems was, and still is, scalability. In early attempts, even one-step successor computations where exponential in the number of continuous variables, so there were obvious limits to applying reachability to systems of practical interest. Thanks in particular to the breakthroughs by Le Guernic and Girard in 2005 and 2008, the one-step successor computation now scale easily to hundreds of variables, with room for further improvements through parallelization, model order reduction, and abstraction, etc. Unfortunately, this does not immediately translate to scalable verification approaches. Chaining the one-step successor operations can lead to an explosive growth in both the approximation error and the number of sets that are produced. The refinement techniques in Chapters 4 and 6 are a first step towards a remedy for both difficulties, since they reduce the error and the number of sets without necessarily hurting scalability. The optimal clustering techniques presented in Sect. 5 can help to master the number of sets without compromising accuracy. Nonetheless, further research and engineering

is required to push refinement and clustering to industrial-strength levels, e.g., through merging of neighboring state sets.

One the way to industrial applications, there are two other difficulties apart from scalability. The construction of suitable verification models is a first and major hurdle. We are supporting this process with a semi-automatic translation tool that converts commercial simulation models into networks hybrid automata [75], and extending analysis techniques that reflect the semantics of such models [74]. Industrial simulation models need to be simplified on the one hand, and on the other hand enriched, e.g., through nondeterministic operating conditions, in order to be able to take advantage of set-based reachability. The second hurdle towards applications is the identification and specification of suitable properties. In our experience, design requirements do not always meet the paradigm of a forbidden set of bad states. Typical specifications can often be encoded as reachability problems by using so-called observer or monitor-automata [54], but this is not necessarily a trivial process for timed and hybrid properties, and merits further study. Other specifications, e.g., stable oscillations, can be handled by combining proofs with monitor automata [43]. Note that for prototypic properties the proof only needs to be carried out once per monitor. The monitor can then be applied to arbitrary systems under test without further need for proofs. Another way of specifying correct or desired behavior may be through performance measures. New criteria may be developed that are better suited to reachability than those typically used in control systems. As an example, consider linear quadratic integrals, which have desirable theoretic properties, but introduce nonlinearities that are not easy to handle for reachability tools. Integrating the absolute value instead may be easier since it is a piecewise linear function, which falls precisely into the class for which we presented efficient and scalable methods.

Bibliography

[1] M. Althoff. An introduction to CORA 2015. In G. Frehse and M. Althoff, editors, *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, volume 34 of *EPiC Series in Computer Science*, pages 120–151. EasyChair, 2015.

[2] M. Althoff, C. L. Guernic, and B. H. Krogh. Reachable set computation for uncertain time-varying linear systems. In Caccamo et al. [22], pages 93–102.

[3] M. Althoff and B. H. Krogh. Avoiding geometric intersection operations in reachability analysis of hybrid systems. In *Hybrid Systems: Computation and Control (HSCC'12)*, pages 45–54. ACM, 2012.

[4] 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.

[5] R. Alur. Formal verification of hybrid systems. In S. Chakraborty, A. Jerraya, S. K. Baruah, and S. Fischmeister, editors, *EMSOFT*, pages 273–278. ACM, 2011.

[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(1):3–34, 1995.

[7] R. Alur, T. Dang, and F. Ivancic. Counterexample-guided predicate abstraction of hybrid systems. *Theor. Comput. Sci.*, 354(2):250–271, 2006.

[8] E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Inf.*, 43(7):451–476, 2007.

[9] E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In E. Brinksma and K. G. Larsen, editors, *CAV*, volume 2404 of *LNCS*, pages 365–370. Springer, 2002.

[10] E. Asarin, T. Dang, O. Maler, and O. Bournez. Approximate reachability analysis of piecewise-linear dynamical systems. In *HSCC'00*, LNCS. Springer, 2000.

[11] E. Asarin, T. Dang, O. Maler, and R. Testylier. Using redundant constraints for refinement. In *Automated Technology for Verification and Analysis*, pages 37–51. Springer, 2010.

[12] 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.

[13] R. Baier and F. Lempio. Computing aumann's integral. *Progress in Systems and Control Theory*, 18:71–71, 1994.

[14] S. Bak. Hycreate: A tool for overapproximating reachability of hybrid automata. `stanleybak.com/projects/hycreate`, 2012.

[15] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar. *Convex Analysis and Optimization*. Athena Scientific, 2003.

[16] S. Bogomolov, A. Donzé, G. Frehse, R. Grosu, T. T. Johnson, H. Ladan, A. Podelski, and M. Wehrle. Abstraction-based guided search for hybrid systems. In E. Bartocci and C. R. Ramakrishnan, editors, *Model Checking Software - 20th International Symposium, SPIN 2013, Stony Brook, NY, USA, July 8-9, 2013. Proceedings*, volume 7976 of *Lecture Notes in Computer Science*, pages 117–134. Springer, 2013.

[17] S. Bogomolov, A. Donzé, G. Frehse, R. Grosu, T. T. Johnson, H. Ladan, A. Podelski, and M. Wehrle. Guided search for hybrid systems based on coarse-grained space abstractions. *International Journal on Software Tools for Technology Transfer*, pages 1–19, 2015.

[18] S. Bogomolov, G. Frehse, M. Greitschus, R. Grosu, C. S. Pasareanu, A. Podelski, and T. Strump. Assume-guarantee abstraction refinement meets hybrid systems. In E. Yahav, editor, *Hardware and Software: Verification and Testing - 10th International Haifa Verification Conference, HVC 2014, Haifa, Israel, November 18-20, 2014. Proceedings*, volume 8855 of *Lecture Notes in Computer Science*, pages 116–131. Springer, 2014.

[19] S. Bogomolov, G. Frehse, R. Grosu, H. Ladan, A. Podelski, and M. Wehrle. A box-based distance between regions for guiding the reachability analysis of spaceex. In P. Madhusudan and S. A. Seshia, editors, *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 479–494. Springer, 2012.

[20] D. Brück, H. Elmqvist, S. E. Mattsson, and H. Olsson. Dymola for multi-engineering modeling and simulation. In *Proceedings of Modelica*, 2002.

[21] R. E. Burkard, H. W. Hamacher, and G. Rote. Sandwich approximation of univariate convex functions with an application to separable convex programming. *Naval Res. Logistics*, 38:911–924, 1991.

[22] M. Caccamo, E. Frazzoli, and R. Grosu, editors. *Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2011, Chicago, IL, USA, April 12-14, 2011*. ACM, 2011.

[23] X. Chen, E. Abraham, and G. Frehse. Efficient bounded reachability computation for rectangular automata. In I. Potapov and G. Delzanno, editors, *Reachability Problems*, volume 6945 of *LNCS*, pages 139–152. Springer, 2011.

[24] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *RTSS*, pages 183–192. IEEE Computer Society, 2012.

[25] X. Chen, S. Schupp, I. B. Makhlouf, E. Ábrahám, G. Frehse, and S. Kowalewski. A benchmark suite for hybrid systems reachability analysis. In K. Havelund, G. J. Holzmann, and R. Joshi, editors, *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*, volume 9058 of *Lecture Notes in Computer Science*, pages 408–414. Springer, 2015.

[26] A. Chutinan and B. H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *HSCC'99*, LNCS, pages 76–90. Springer, 1999.

[27] A. Chutinan and B. H. Krogh. Computational techniques for hybrid system verification. *IEEE Trans. Automat. Contr.*, 48(1):64–75, 2003.

[28] S. Cotton, G. Frehse, and O. Lebeltel. The spaceex modeling language. `spaceex.imag.fr/documentation/user-documentation/`, dec 2010.

[29] W. Damm, S. Disch, H. Hungar, S. Jacobs, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz. Exact state set representations in the verification of linear hybrid systems with large discrete state space. In *ATVA*, 2007.

[30] T. Dang, T. Dreossi, and C. Piazza. Parameter synthesis using parallelotopic enclosure and applications to epidemic models. In *Int. Ws. Hybrid Systems and Biology HSB'14*, LNBI. Springer, 2014.

[31] T. Dang, O. Maler, and R. Testylier. Accurate hybridization of nonlinear systems. In K. H. Johansson and W. Yi, editors, *HSCC*, pages 11–20. ACM, 2010.

[32] G. B. Dantzig and T. M. N. *Linear Programming 2: Theory and Extensions.* Springer, 2003.

[33] A. Donzé and G. Frehse. Modular, hierarchical models of control systems in spaceex. In *Control Conference (ECC), 2013 European*, pages 4244–4251. IEEE, 2013.

[34] P. S. Duggirala and A. Tiwari. Safety verification for linear systems. In *EMSOFT'13*, pages 1–10. IEEE, 2013.

[35] G. Frehse. PHAVer: algorithmic verification of hybrid systems past HyTech. *STTT*, 10(3):263–279, 2008.

[36] G. Frehse. An introduction to spaceex. `spaceex.imag.fr/documentation/user-documentation/`, dec 2010.

[37] G. Frehse. SpaceEx state space explorer. Verimag, Grenoble, `http://spaceex.imag.fr`, 2010.

[38] G. Frehse. Reachability of hybrid systems in space-time. In *Embedded Software (EMSOFT), 2015 International Conference on*, pages 41–50, Oct 2015.

[39] G. Frehse, S. Bogomolov, M. Greitschus, T. Strump, and A. Podelski. Eliminating spurious transitions in reachability with support functions. In A. Girard and S. Sankaranarayanan, editors, *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC'15, Seattle, WA, USA, April 14-16, 2015*, pages 149–158. ACM, 2015.

[40] G. Frehse, A. Hamann, S. Quinton, and M. Wöhrle. Formal Analysis of Timing Effects on Closed-loop Properties of Control Software. In *35th IEEE Real-Time Systems Symposium 2014 (RTSS)*, Rome, Italy, Dec. 2014.

[41] G. Frehse, S. K. Jha, and B. H. Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In M. Egerstedt and B. Mishra, editors, *HSCC*, volume 4981 of *LNCS*, pages 187–200. Springer, 2008.

[42] G. Frehse, R. Kateja, and C. 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.

[43] G. Frehse, B. H. Krogh, and R. A. Rutenbar. Verifying analog oscillator circuits using forward/backward abstraction refinement. In G. G. E. Gielen, editor, *DATE*, pages 257–262, 2006.

[44] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In G. Gopalakrishnan and S. Qadeer, editors, *CAV*, volume 6806 of *LNCS*, pages 379–395. Springer, 2011.

[45] G. Frehse and R. Ray. Design principles for an extendable verification tool for hybrid systems. In *ADHS'09 : 3rd IFAC Conference on Analysis and Design of Hybrid Systems*, 2009.

[46] G. Frehse and R. Ray. Flowpipe-guard intersection for reachability computations with support functions. In *IFAC ADHS*, pages 94–101, 2012.

[47] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[48] P. K. Ghosh and K. V. Kumar. Support function representation of convex bodies, its application in geometric computing, and some related representations. *Comput. Vis. Image Underst.*, 72(3):379–403, Dec. 1998.

[49] A. Girard. Reachability of uncertain linear systems using zonotopes. In M. Morari and L. Thiele, editors, *HSCC*, volume 3414 of *LNCS*, pages 291–305. Springer, 2005.

[50] A. Girard and C. Le Guernic. Efficient reachability analysis for linear systems using support functions. In *IFAC World Congress*, 2008.

[51] A. Girard, C. Le Guernic, and O. Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In *HSCC'06*. Springer, 2006.

[52] T. J. Graettinger and B. H. Krogh. Hyperplane method for reachable state estimation for linear time-invariant systems. *Journal of Optimization Theory and Applications*, 69(3):555–588, 1991.

[53] M. R. Greenstreet. Verifying safety properties of differential equations. In *Computer Aided Verification*, pages 277–287. Springer, 1996.

[54] N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In *Algebraic Methodology and Software Technology (AMAST'93)*, pages 83–96. Springer, 1994.

[55] T. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[56] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.

[57] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.

[58] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

[59] W. Kühn. Rigorously computed orbits of dynamical systems without the wrapping effect. *Computing*, 61(1):47–67, 1998.

[60] A. Kurzhanski and P. Varaiya. Reachability analysis for uncertain systems—the ellipsoidal technique. *Dynamics of Continuous, Discrete and Impulsive Systems Series B: Applications and Algorithms*, 9(3b):347–367, 2002.

[61] A. B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In N. A. Lynch and B. H. Krogh, editors, *Hybrid Systems: Computation and Control, Third International Workshop, HSCC 2000, Pittsburgh, PA, USA, March 23-25, 2000, Proceedings*, volume 1790 of *Lecture Notes in Computer Science*, pages 202–214. Springer, 2000.

[62] A. B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability under state constraints. *SIAM J. Control and Optimization*, 45(4):1369–1394, 2006.

[63] A. B. Kurzhanski and P. Varaiya. *Dynamics and Control of Trajectory Tubes*. Springer, 2014.

[64] A. A. Kurzhanskiy and P. Varaiya. Ellipsoidal toolbox (et). In *Decision and Control, 2006 45th IEEE Conference on*, pages 1498–1503. IEEE, 2006.

[65] C. Le Guernic. *Reachability analysis of hybrid systems with linear continuous dynamics*. PhD thesis, Université Grenoble 1 - Joseph Fourier, 2009.

[66] C. Le Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *LNCS*, pages 540–554. Springer, 2009.

[67] C. Le Guernic and A. Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250 – 262, 2010. IFAC World Congress 2008.

[68] A. V. Lotov, V. A. Bushenkov, and G. K. Kamenev. *Interactive Decision Maps*, volume 89 of *Applied Optimization*. Kluwer, 2004.

[69] F. Maffray. On the coloration of perfect graphs. In B. A. Reed and C. L. Sales, editors, *Recent Advances in Algorithms and Combinatorics*, CMS Books in Mathematics, pages 65–84. Springer, 2003.

[70] O. Maler. Algorithmic verification of continuous and hybrid systems. In *Int. Workshop on Verification of Infinite-State System (Infinity)*, 2013.

[71] MapleSoft. Maplesim 7: Advanced system-level modeling. `http://www.maplesoft.com/products/maplesim`, 2015.

[72] MathWorks. Mathworks simulink: Simulation et model-based design, Mar. 2014. `www.mathworks.fr/products/simulink`.

[73] S. Minopoli and G. Frehse. Non-convex invariants and urgency conditions on linear hybrid automata. In A. Legay and M. Bozga, editors, *Formal Modeling and Analysis of Timed Systems - 12th International Conference, FORMATS 2014, Florence, Italy, September 8-10, 2014. Proceedings*, volume 8711 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2014.

[74] S. Minopoli and G. Frehse. From simulation models to hybrid automata using urgency and relaxation. In *HSCC'16*, 2016 (to appear).

[75] S. Minopoli and G. Frehse. Sl2sx translator: From simulink to spaceex verification tool. In *HSCC'16*, 2016 (to appear).

[76] I. M. Mitchell. Comparing forward and backward reachability as tools for safety analysis. In *HSCC'07*, pages 428–443, 2007.

[77] A. Pereira and M. Althoff. Safety control of robots under computed torque control using reachable sets. In *IEEE Int. Conf. Robotics and Automation*, 2015.

[78] P. Prabhakar and M. Viswanathan. A dynamic algorithm for approximate flow computations. In Caccamo et al. [22], pages 133–142.

[79] R. Ray and G. Frehse. An approach to direct minimization of convex piecewise linear functions. Technical report, Verimag, Nov. 2011.

[80] C. Scholl, S. Disch, F. Pigorsch, and S. Kupferschmid. Computing optimized representations for non-convex polyhedra by detection and removal of redundant linear constraints. In *TACAS*, 2009.

[81] S. Schupp, E. Ábrahám, X. Chen, I. B. Makhlouf, G. Frehse, S. Sankaranarayanan, and S. Kowalewski. Current challenges in the verification of hybrid systems. In C. Berger and M. R. Mousavi, editors, *Cyber Physical Systems. Design, Modeling, and Evaluation - 5th International Workshop, CyPhy 2015, Amsterdam, The Netherlands, October 8, 2015, Proceedings*, volume 9361 of *Lecture Notes in Computer Science*, pages 8–24. Springer, 2015.

[82] S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control: Analysis and Design*. John Wiley & Sons, 2005.

[83] P. Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.

[84] H. Tiwary. On the hardness of computing intersection, union and minkowski sum of polytopes. *Discrete & Computational Geometry*, 40(3):469–479, 2008.

[85] UnCoVerCPS. EU H2020 project on unifying control and verification of cyber-physical systems (uncovercps), grant number 643921. `cps-vo.org/group/UnCoVerCPS`, 2015.

[86] G. van den Bergen. *Collision detection in interactive 3D computer animation*. PhD thesis, Eindhoven University of Technology, 1999.

[87] P. Varaiya. Reach set computation using optimal control. In *Proc. KIT Workshop*, pages 377–383, 1997.