# Automatic Time-Unbounded Reachability Analysis of Hybrid Systems

by

**Mirco Giacobbe**

September 22, 2019

*A thesis presented to the*
*Graduate School*
*of the*
*Institute of Science and Technology Austria, Klosterneuburg, Austria*
*in partial fulfillment of the requirements*
*for the degree of*
*Doctor of Philosophy*

IST AUSTRIA
*Institute of Science and Technology*

The thesis of Mirco Giacobbe, titled *Automatic Time-Unbounded Reachability Analysis of Hybrid Systems*, is approved by:

**Supervisor**: Thomas A. Henzinger, IST Austria, Klosterneuburg, Austria

Signature: ________________________

**Committee Member**: Goran Frehse, ENSTA ParisTech, Palaiseau, France

Signature: ________________________

**Committee Member**: Radu Grosu, TU Wien, Vienna, Austria

Signature: ________________________

**Defense Chair**: Daria Siekhaus, IST Austria, Klosterneuburg, Austria

Signature: ________________________

signed page is on file

IST Austria Thesis, ISSN: 2663-337X

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: ___________________________

Mirco Giacobbe

September 22, 2019

signed page is on file

# Abstract

Hybrid automata combine finite automata and dynamical systems, and model the interaction of digital with physical systems. Formal analysis that can guarantee the safety of all behaviors or rigorously witness failures, while unsolvable in general, has been tackled algorithmically using, e.g., abstraction, bounded model-checking, assisted theorem proving. Nevertheless, very few methods have addressed the time-unbounded reachability analysis of hybrid automata and, for current sound and automatic tools, scalability remains critical.

We develop methods for the polyhedral abstraction of hybrid automata, which construct coarse overapproximations and tightens them incrementally, in a CEGAR fashion. We use template polyhedra, i.e., polyhedra whose facets are normal to a given set of directions. While, previously, directions were given by the user, we introduce (1) the first method for computing template directions from spurious counterexamples, so as to generalize and eliminate them. The method applies naturally to convex hybrid automata, i.e., hybrid automata with (possibly non-linear) convex constraints on derivatives only, while for linear ODE requires further abstraction. Specifically, we introduce (2) the conic abstractions, which, partitioning the state space into appropriate (possibly non-uniform) cones, divide curvy trajectories into relatively straight sections, suitable for polyhedral abstractions. Finally, we introduce (3) space-time interpolation, which, combining interval arithmetic and template refinement, computes appropriate (possibly non-uniform) time partitioning and template directions along spurious trajectories, so as to eliminate them.

We obtain sound and automatic methods for the reachability analysis over dense and unbounded time of convex hybrid automata and hybrid automata with linear ODE. We build prototype tools and compare—favorably—our methods against the respective state-of-the-art tools, on several benchmarks.

# Acknowledgments

I thank my supervisor Tom for guiding me through my Phd and for encouraging me to pursue the problems I wanted to; I thank him for teaching me how to write papers, when it worked, and for his patience, when it didn't. I thank Tom for all inspiring ideas, and for the example.

I thank Goran for getting his hands dirty. I thank him for sharing with me problems to solve and for bouncing ideas off each other, for all technical insights and for checking every single line of the papers we wrote together. Also, I thank Goran for his hospitality, when having me stay in Grenoble and Paris.

I thank Sergiy for introducing me to the problem domain of this thesis and I thank Hui for his invaluable help and knowledge in developing the conic abstractions. I thank both for their energy and for sharing with me their expertise.

I thank Ashutosh, Calin, Tanja, and Tiago for jumping with me into the model-checking of gene regulatory networks. I thank them for inspiring me and all help they gave to me in the early stage of my Phd.

I thank all my fellow group members and all group members of Chatterjee's group for the good time we had together and for sharing with me, every day, a bit of their talent.

I finally thank my family, for their love and support.

## About the Author

Mirco Giacobbe completed a BSc in Computer Science at the University of Trento, a MSc is Software Systems Engineering at RWTH Aachen, and a MSc in Computer Science at the University of Trento. He is currently a Phd student in Henzinger's group, at IST Austria. His main research interests include the analysis of timed and hybrid systems, and the application of formal methods in systems biology and artificial intelligence.

# List of Publications

Chapter 4 is a joint work with Sergiy Bogomolov, Goran Frehse, and Thomas A. Henzinger, and appeared in the proceedings of TACAS 2017 in the paper *Counterexample-guided refinement of template polyhedra* [23]; its theoretical foundations are presented in Ch. 3. Chapter 5 is a joint work with Sergiy Bogomolov, Hui Kong, and Thomas A. Henzinger, and appeared in the paper *Conic Abstractions for Hybrid Systems* [24], in the proceedings of FORMATS 2017. Chapter 6 is a joint work with Goran Frehse and Thomas A. Henzinger, and appeared in the paper *Space-Time Interpolants* [55], in the proceedings of CAV 2018.

# Table of Contents

**Abstract** **v**

**Acknowledgments** **vi**

**About the Author** **vii**

**List of Publications** **viii**

**List of Tables** **xi**

**List of Figures** **xii**

**List of Abbreviations** **xiv**

**1 Introduction** **1**

**2 Template-polyhedral Abstraction of Hybrid Automata** **10**

2.1 Hybrid Automata . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.2 Template-polyhedral Abstraction . . . . . . . . . . . . . . . . . . . . . . 14

2.3 Support Functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

**3 Automatic Template Refinement** **19**

3.1 Halfspace Interpolation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

3.2 Interpolation as Convex Optimization . . . . . . . . . . . . . . . . . . . . 28

**4 Counterexample-guided Refinement for Convex Hybrid Automata 34**

4.1 Motivating Example . . . . . . . . . . . . . . . . . . . . . . . . . . . 36

4.2 Abstractions of Convex Hybrid Automata using Template Polyhedra . . . 38

4.3 Counterexample-guided Refinement using Halfspace Interpolants . . . . . . 41

4.4 Craig's Interpolation as Convex Optimization . . . . . . . . . . . . . . . . 45

4.5 Abstraction Refinement for Quadratic Systems . . . . . . . . . . . . . . . . 51

4.6 Benchmarks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53

4.7 Experimental Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . 59

**5 Conic Abstractions for Affine Hybrid Automata 63**

5.1 Affine Systems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65

5.2 Conic Abstractions of Affine Systems . . . . . . . . . . . . . . . . . . . 66

5.3 Diagonalizable Systems . . . . . . . . . . . . . . . . . . . . . . . . . . 72

5.4 Time-unbounded Reachability Analysis . . . . . . . . . . . . . . . . . . . 75

5.5 Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 79

**6 Space-Time Interpolation for Affine Hybrid Automata 82**

6.1 Motivating Example . . . . . . . . . . . . . . . . . . . . . . . . . . . 85

6.2 Hybrid Automata with Piecewise Affine Dynamics . . . . . . . . . . . . . 87

6.3 Time Abstraction using Interval Arithmetic . . . . . . . . . . . . . . . . 88

6.4 Space Abstraction using Support Functions . . . . . . . . . . . . . . . . 93

6.5 Abstraction Refinement using Space-time Interpolants . . . . . . . . . . . 95

6.6 Experimental Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . 100

6.7 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102

**7 Conclusions 103**

**Bibliography 107**

# List of Tables

4.1 Runtimes for the reachability of convex hybrid automata. . . . . . . . . . . 60

5.1 Runtimes for the conic abstraction of affine hybrid automata . . . . . . . . 80

6.1 Runtimes for the reachability of affine hybrid automta using space-time interpolants . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100

# List of Figures

2.1 Counterexample-guided abstraction refinement loop. . . . . . . . . . . . . . . 11

2.2 Thermostat automaton of an air conditioner. . . . . . . . . . . . . . . . . . . 14

2.3 A discrete time LTI system. . . . . . . . . . . . . . . . . . . . . . . . . . . 15

2.4 A sequence of template polyhedral abstractions. . . . . . . . . . . . . . . . . 16

3.1 Template refinement using halfspace interpolants. . . . . . . . . . . . . . . 20

3.2 Intersection that does not allow proper halfspace interpolation. . . . . . . . 21

3.3 Intersection that does not allow strong halfspace interpolation. . . . . . . . 24

3.4 Intersection of halfspaces for which Eq. 3.18 is incomplete. . . . . . . . . . 31

4.1 A convex hybrid automaton . . . . . . . . . . . . . . . . . . . . . . . . . . 36

4.2 Template-polyhedra abstraction refinement of a CHA . . . . . . . . . . . . 37

4.3 Halfspace interpolants for a spurious path . . . . . . . . . . . . . . . . . . 43

4.4 Process automaton of the Fischer's protocol . . . . . . . . . . . . . . . . . 54

4.5 Follower automaton of the adaptive cruise controller . . . . . . . . . . . . . 55

4.6 Compression master of the TTEthernet protocol . . . . . . . . . . . . . . . 57

4.7 Synchronization master of the TTEthernet protocol . . . . . . . . . . . . . 57

5.1 Conic partitioning of state and derivative space . . . . . . . . . . . . . . . 67

5.2 Accuracy of conic abstractions . . . . . . . . . . . . . . . . . . . . . . . . 69

5.3 State vs. derivative space conic partitioning . . . . . . . . . . . . . . . . . 71

5.4 Conic abstractions with different precicions . . . . . . . . . . . . . . . . . 78

5.5 Run-times for the conic abstraction of purely continuous systems . . . . . . 79

5.6 Conic abstractions of the heating benchmark . . . . . . . . . . . . . . . . . . 81

6.1 Space-only vs. Time-only vs. Space-time abstraction refinement . . . . . . 86

# List of Abbreviations

| | |
|---|---|
| **CHA** | Convex Hybrid Automaton |
| **LHA** | Linear Hybrid Automaton |
| **ODE** | Ordinary Differential Equation |
| **PWA** | Piece-wise Affine Hybrid Automaton |
| **QHA** | Quadratic Hybrid Automaton |

# 1 Introduction

Hybrid systems arise from the interaction of digital—discrete—systems and physical—continuous—systems. Typical examples are digitally controlled physical plants, from the autopilot of an airplane to the thermostat of your home. Other examples are digital systems subject to physical inputs, from sophisticated measurement devices to audio card of your laptop; in a sense, every real-time system is so, if one considers time as a continuous physical quantity. As hybrid systems are often employed in critical applications, quality assurance is necessary. Unfortunately analyzing hybrid systems at runtime is even harder than analyzing programs, as their behavior depends on a physical environment; one does not need to resort to Mars rovers, but just imagine how onerous it would be to test the implementation of a thermostat across several conditions. For this reason, hybrid systems are preferably analyzed using models. The digital component consists of a computer program, which is modeled by means of an automaton that manipulates integer numbers. The physical component, in a large variety of cases, is modeled by means of differential equations or, more generally, flows over the reals. The combination of automata and flows gives rise to hybrid automata [92; 7; 63]. More precisely, a hybrid automaton models a family of continuous trajectories—the flowpipe—whose initial and flow conditions are non-deterministically governed by an automaton. In its turn, the hybrid automaton non-deterministically induces discontinuous switches. Our subject of study is their safety verification of hybrid automata.

Verifying safety amounts to deciding whether all systems' trajectories remain within a safe region of states; dually, it consists of proving that no trajectory can reach an unsafe region and, for this reason, we talk about reachability analysis. Formal decidability, that is the existence of a procedure that is sound, in the sense that if it answers safe or unsafe then the system is so, and complete, in the sense that it always terminates providing a definite answer, has been identified for specific classes of hybrid automata. For timed automata, i.e., hybrid automata whose variables have equal and constant derivatives, and guards and invariant are rectangular, reachability is decidable [11]; on the other hand, it becomes undecidable if one allows two variables to have different constant derivatives [71]. Only if one initializes the variables before every derivative change, and never compares variables with different derivatives, then reachability is decidable; indeed, under the same conditions, it is decidable for rectangular hybrid automata, i.e., hybrid automata where derivatives are taken non-deterministically from rectangular regions [71]. Beyond initialized rectangular automata, all general reachability problems for hybrid automata are undecidable, with only a few exceptions, e.g., o-minimal hybrid automata [84; 83], or orbit problems [77; 60; 32; 38; 39]. In particular, it is undecidable for the simplest of our classes of study, the linear hybrid automata (LHA), i.e., hybrid automata with non-deterministic derivatives taken within polyhedra. Nevertheless, one can get around undecidability by relaxing on soundness or completeness, but, in this case, safety and reachability are to be treated distinctly. For instance, take overapproximative abstraction [46], sound for safety, but cannot claim a bad state reachable, or take time-bounded analysis [29], sound for reachability, but cannot declare the system safe. Also, one can achieve one-sided completeness by, e.g., incrementally tightening an abstraction [97; 45], terminating for systems that are robustly safe (i.e., safe with some gap), not terminating otherwise, or incrementally increasing a time-bound, terminating for systems that reach the bad region [22], not necessarily terminating for safe systems. In this work, we develop a portfolio of techniques all of which, for safety, are sound but formally incomplete. Nevertheless, we experiment and demonstrate they terminate in practice, on reference benchmarks.

The reachability analysis—in practice—has been studied both the by control theory and formal methods communities. Classically, control theorists focus on the analysis of purely continuous, but highly non-linear systems, seen as transformers of signals. They use analytical proofs or, alternatively, numerical methods. In particular, numerical methods often suffice when the robustness can be guaranteed, in the sense that small perturbations in the input signal lead to small perturbations to the output signal. Unfortunately, hybrid controllers are often highly sensitive to perturbations, even for systems with relatively simple continuous dynamics, such as linear ODE; this is due to the fact that they are highly discontinuous. On the other hand, their complex discrete structure and relatively simple continuous dynamics make them amenable to algorithmic analysis tools, inspired by the verification of computer programs.

Formal methods for the reachability analysis of hybrid systems has been studied from multiple perspectives, mainly inspired by the theories of abstract interpretation [46; 67], SMT-solving [18], and theorem proving. Abstractly interpreting a system consists of computing an abstract representation of its reachable states—the reach set—and prove safety by showing disjointedness from a set of undesired states—the bad set. The efficiency and precision of abstract interpreters are mainly characterized by the data structure used for the representation of state sets. The first abstract interpreter for hybrid automata, Hytech [66; 68], used polyhedra and treated systems with linear reach sets [6], i.e., linear hybrid automata (LHA); as for systems with linear ODE, i.e., piece-wise affine automata (PWA), whose reach sets are generally non-linear, linear phase approximation into LHA was applied [69]. Polyhedral methods have been further optimized by HybridSAL, which employs predicate abstraction [8], and Phaver [52], which employs abstraction widening; Phaver remained, until today, the state-of-the-art tool for the automatic time-unbounded reachability of LHA. The direct abstract interpretation of PWA has been tackled by polyhedral approximations of the flow pipe, pioneered in the tool Checkmate [40]. Yet, constructing polyhedral flowpipes is computationally costly, therefore efficient but coarser data structures have been introduced; these include hyperrectangles [27], ellipsoids [82], zonotopes [58], and template polyhedra [102; 86], implemented in d/dt [15], VeriSHIFT [26], CORA [5], SpaceEX [57]. Besides, flow pipe approximation relies on partitioning the continuous time and controlling the approximation error, at the expenses of termination (in practice) for the time-unbounded problem; notably, all flow pipe approximation tools bound the continuous time. Similarly, tools based on validated numerical analysis also bound the continuous time; one example is Flow* [36], which uses interval arithmetic and Taylor models to solve (non-linear) polynomial differential equations at successive time intervals [34]. On the other side of the spectrum, methods based on SMT-solving encode reachability or, similarly, inductive invariants, into first-order logic formulae whose variable account for variables valuations at discrete steps; naturally, all these methods are discrete-time bounded. The expressivity of the solver, coupled with the encoding method, translates into the expressivity of the analyzable automata, e.g., LHA in the tool Bach [30], PWA or hybrid systems with polynomial solution in Hycomp [41], systems with non-linear differential equations, including polynomials with trigonometric and exponential functions, in iSat [50] and dReach [81]. Also, unfortunately, expressivity comes at the price of completeness, both in the sense that safety for the time-bounded does not imply the safety of the time-unbounded problem, but also that a witness from the solver may not correspond to a counterexample for the hybrid system (as for most methods for non-linear systems); for theorem provers, this is not the case. Theorem provers for hybrid systems [94] or, more specifically, Keymera [95] assist the user to produce sound and complete proofs of, e.g., safety for any kind system and for unbounded-time; with user intervention, Keymera can construct an invariant for the reach set like, e.g., a barrier certificate between flow pipe and bad set. Although, the synthesis of barrier certificate and other invariants have been also fully automatized [80], but for small systems.

We perform the reachability analysis of hybrid automata over dense and unbounded time by abstract interpretation. The ingredients of an abstract interpreter are a state transformation that abstracts the elapse of continuous or discrete time—the post operator—and a data structure for the abstract representation of sets of states. We consider post operators that preserve convexity, in the sense that convex sets are necessarily mapped into convex sets, and represent sets of states using template polyhedra. The introduce a theory for the abstraction refinement for template polyhedra which naturally applies to LHA and their generalization involving convex constraints—the convex hybrid automata. We extend our theory to PWA, whose flowpipes are not necessarily convex, introducing a technique for their linear phase approximation—the conic abstractions—and for their flowpipe approximation—the space-time interpolants.

Template polyhedra are convex polyhedra whose defining halfspaces are orthogonal to a template, i.e., a finite set of directions [103; 102]. In other words, they are those conjunctions of linear inequalities where all coefficients are fixed and constants can vary; they naturally generalize geometrical representations like intervals or octagons, yet maintain low computational cost for several set operations; nevertheless, their precision is sensitive to the choice of template. In fact, even computing the tightest of the template polyhedra around a set won't necessarily bring to an exact representation. This holds for linear sets, think about using intervals or octagons for representing arbitrary polyhedra, and for non-linear sets, think about using any finite set of directions for representing ellipses or parabolas. In addition, template polyhedra suffer from the so-called wrapping effect, that is to say that even if you represent initial and guard constraints of a hybrid automaton precisely, discrete transitions and time elapse might make new directions necessary. Think about representing a box using intervals, applying a slight rotation, and representing it again using intervals. Thus the question is: how do you choose the template? We discover template directions from spurious counterexamples and add to the template a few of them at a time. Let us look at a refinement workflow. Initially, we search for a spurious counterexample using a fixed (and possibly empty) template. Once such a counterexample is found, we extract an inductive sequence of halfspace interpolants, i.e., Craig's interpolants that consist of single linear inequalities [2]. We take their outward pointing directions and we add them to the template. Such directions eliminate the counterexample and generalize to all other counterexamples with the same switching sequence and any (and possibly unbounded) time elapse. We repeat the procedure in CEGAR fashion [44].

Discovering directions from counterexamples presents several subtleties. First of all, even for systems given by linear constraints only, directions cannot always be deduced statically by analyzing the constraints [33]; the template directions that are necessary to abstract a reach set in such a way the abstraction is disjoint from the bad set do not necessarily appear in either constraint. The refining directions are related to the normal directions of the halfspaces separating reach and bad set [54]. Nevertheless, a simple local search for separating might not be sufficient; along a spurious counterexample the overapproximation error may be caused by multiple subsequent template polyhedra (this is the wrapping effect), therefore refinement may need to add directions to multiple of these abstractions. Indeed, refining the template directions with respect to a spurious counterexample consists of finding a sequence of halfspaces along the path that inductively depend on one another and that finally prove separation from the bad set of states. We develop a theory for the refinement of template directions: we provide conditions for which a system admits refinement and, also, provide a method for computing the directions. For abstractions whose post operator is defined in terms of Minkowski sums, linear transformation, intersections over convex sets, we show that, under sufficient conditions such as, e.g., compactness or linearity, halfspace interpolants always exist. We also show that computing halfspace interpolants consists of solving a convex program; as a result, any appropriate convex programs solver is a valid template refiner.

Our theory of abstraction refinement for template polyhedra naturally applies to convex hybrid automata. Convex hybrid automata (CHA) are hybrid automata whose dynamics are given by convex guards and invariants and convex constraints on the derivatives only; in other words, the derivative along a trajectory do not depend on the current state, but is non-deterministically taken within that constraint. Multi-variate timed systems whose clock drifts are given by non-linear constraints or probabilistic systems whose inputs are taken from a normal distribution truncated by an ellipsoid are modeled by CHA. The time-unbounded abstraction of CHA can be carried out by a post operator consisting of Minkowski sums, linear transformation, intersections, with the addition of the conical hull of the derivative constraints. We extend our theory with the conical hull operator and, as a result, apply template refinement to any spurious counterexample. As a result, we enable the time-unbounded reachability analysis of convex hybrid automata, which was not practical before using polyhedral methods.

The semantics of convex hybrid automata preserves convexity in the sense that the continuous time evolution of a convex set is a convex set; this is not the case for hybrid automata with linear ODE, whose continuous time evolution is given by an exponential function and, since it induces non-convex sets, does not directly allow halfspace interpolation as for CHA. For this reason, we abstract the system constructing post operators that are convex but overapproximative. First, we partition the system, and then, within every partition, we construct a convex overapproximation of the respective reach set, for which halfspace interpolation applies. We obtain two-level abstraction refinement methods in which the first level takes care of the partitioning and the second level of the template. With this framework in mind, we introduce two methods that are suitable for different kinds of systems: one based on the partitioning of space—the conic abstractions—and a method based on the partitioning of time—the space-time interpolants.

The conic abstractions is a linear phase approximation method which partitions the state space of the PWA system and, within each partition, overapproximates the differential equation with a constraint over derivatives only. More specifically, it substitutes the differential equation with the set of all derivatives that can occur within the partition; since it partitions the system into convex sets, this induces a convex hybrid automaton, which we analyze using template polyhedra as above. Together with the template, the precision of the overapproximation is entirely determined by the partitions. We partition the state space into cones ensuring, as we show, the set of trajectories within each cone to be as straight as possible, with the result of obtaining a good approximation using linear constraints. In addition, we control the relative angle between any two trajectories, which we call the twisting, by controlling the aperture of every conic partition. Altogether, we obtain an abstraction refinement framework, whose precision is controlled by the value of twisting. Besides, we show that, for diagonalizable systems, the conic abstraction of each time-unbounded flowpipe is guaranteed to terminate.

The space-time interpolants is a flowpipe approximation method which partitions the time into intervals and approximates the solution of the differential equation within each interval. In particular, it approximates the solution with respect to a dense interval of time with a dense family of linear transformations, represented by an interval matrix that we compute using interval arithmetic. We extend our theory of abstraction refinement for template polyhedra with the operation of transformation through an interval matrix. As a result, for each time interval, we induce a convex system for which, again, halfspace interpolation applies. The exploration of the state space terminates when the elapse of time creates a cycle, which we prove by checking inclusion between template polyhedra. In particular, the method benefits from using as-small-as-possible templates for which, as we show on multiple examples, the fixpoint can be found (in practice).

The performance of a time-unbounded abstract interpreter is determined by whether and when the fixpoint is detected which, as our experiments show, benefit from using small templates, whose directions are discovered on demand. The analysis of CHA, the conic abstraction, and space-time interpolation of PWA are fundamentally different, in this respect. The first deals with systems whose dynamics are straight, the second with systems whose dynamics are acyclic (diagonalizable), while the third benefits from cyclic dynamics. Altogether, this work presents a portfolio of techniques for time-unbounded reachability of hybrid systems, each of which is suitable to specific kinds of dynamics, which we evaluated experimentally. We implemented our template refinement method for convex systems with linear and quadratic constraints, obtaining, for the first time, effective time-unbounded reachability analysis of quadratic hybrid automata (QHA), and superior performance for LHA with respect to Phaver; in particular, we evaluated our method on linear and quadratic variants of Fischer's protocol [85], the TTEthernet synchronization protocol [25], and an adaptive cruise controller [73]. Building on the same framework, we demonstrated the efficacy of conic abstraction, on the analysis of a coordinated system of room heaters, and of space-time interpolants, on the analysis of a filtered oscillator and the control rod of a nuclear reactor. Our method outperformed state-of-the-art tools like Phaver, SpaceEx, and Ariadne, which, for most examples, did not terminate within reasonable time for the time-unbounded problem.

At high level, we summarize the contributions of this thesis in the following points.

- We developed a theory for the refinement from the spurious counterexamples from a CEGAR loop, for template polyhedral abstractions; our theory is complete, in the sense that it eliminates any counterexamples of a convex system (Sec. 3 and 4).
- We introduced a linear phase approximation method for PWA, the conic abstractions, which partitions flowpipes into as-straight-as-possible unbounded pieces (Sec. 5).
- We introduced a flowpipe approximation method for PWA, the space-time interpolants, which couples template polyhedra refinement with interval arithmetic (Sec. 6).

On the technical side, we made the following contributions.

- We phrased template refinement as the halfspace interpolation problem and showed that for abstractions consisting of Minkowski sums, linear maps, and intersections of linear or compact sets, refinement is complete (Sec. 3.1) and computable using convex optimization (Sec. 3.2).
- We showed that, to obtain straight linear phase approximations, it is sufficient to partition the invariant into a specific set of cones (Sec. 5.2), and that computing the respective flowpipe terminates for purely continuous diagonalizable systems (Sec. 5.3).

- We extended our theory of template refinement and, with it, the theory of support functions–based abstraction, with the operations of conical hull (Sec. 4.4) and map trough interval matrix (Sec. 6.4).

We have built prototype tools and demonstrated, through multiple benchmarks, the effectiveness of our methods against the state-of-the-art tools for the respective kind of systems, where available. We obtained, for the first time, effective time-unbounded reachability of QHA (Sec. 4.7), and, with respect to the respective state-of-the-art tools, we obtained superior performance for LHA (Sec. 4.7), for the analysis of systems of room heaters (which are diagonalizable PWA), using conic abstractions (Sec. 5.5), for the analysis of oscillators and control rods (which are PWA with cyclic behavior), using space-time interpolants (Sec. 6.6).

We give the preliminary notions of hybrid automata, support functions, and template polyhedra in Sec. 2, and we introduce our theory of refinement for template polyhedra, general conditions for completeness, and an encoding into convex optimization in Sec. 3. In Sec. 4, we extend our theory to conical hulls and instantiate it in a CEGAR loop for the reachability analysis convex hybrid automata; in Sec. 5, we introduce theory and experimental results for the conic abstractions; in Sec. 6, we extend our theory to interval matrices and present algorithm and experimental results for the space-time interpolants.

# 2 Template-polyhedral Abstraction of Hybrid Automata

We target the time-unbounded reachability analysis of hybrid automata. The problem, which we define in Sec. 2.1, does not admit a complete algorithm in the general case [], and for the classes we treat, i.e., linear hybrid automata in Ch. 4, piece-wise affine automata in Ch. 5 and 6. Nevertheless, several techniques for their symbolic analysis have been developed, among which we study, in Sec. 2.2, the abstract interpretation using template polyhedra. We construct an abstractor which takes a hybrid automaton, a bad state, an abstraction precision, i.e., a template, and tells whether the automaton is safe, in which case the automaton is indeed safe, or returns a counterexample. The counterexample may or may not be a genuine counterexample, hence one cannot, directly, tell the system unsafe. We overcome this by a counterexample-guided abstraction refinement (CEGAR) loop [44],

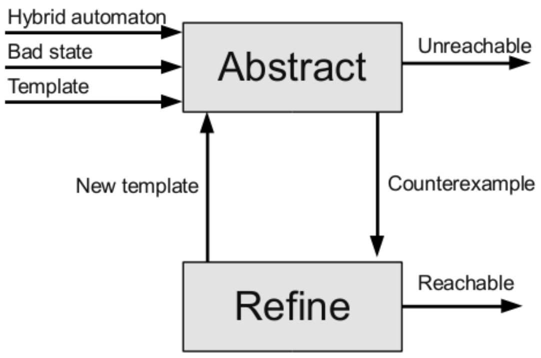


**Figure 2.1: Counterexample-guided abstraction refinement loop.**

as illustrated in Fig. 2.1: we give the counterexample to a refiner, which either tells it genuine, i.e., the bad state reachable, or provides a new template for the abstractor, which eliminates the counterexample. We develop a general theory for the template refinement task, in Ch. 3, and give sufficient conditions for the completeness of deciding whether a given counterexample is genuine. Besides, both abstraction and refinement leverage the support function representation for convex sets [59], which we present in Sec. 2.3.

## 2.1 Hybrid Automata

Hybrid systems combine continuous dynamical systems and discrete models of computation. The continuous fragment is typically modeled by differential equations, or their generalizations into differential algebraic equations, differential inclusions, while the discrete fragment is typically modeled by, e.g., automata, computer programs, process algebra. The combinations of these continuous and discrete formalisms gives rise to, e.g., hybrid automata, with different continuous dynamics and different composition rules [108; 88; 92; 7; 63; 87], hybrid programs [12; 93], hybrid process algebras [47; 21]. Hybrid automata describe piece-wise continuous trajectories over a given set of real variables. In particular, hybrid automata consist of a finite sets of control modes, each of which determines the continuous dynamic of a trajectory, and control switches, which determine the discontinuous transitions between modes. Modes and switches are additionally characterized by invariant and guard conditions over the variables, which respectively determine when a trajectory can enter and stay in a mode and when a switch can be taken. Hybrid automata also offer interfaces for the synchronization of switches between multiple agents (i.e., multiple hybrid automata), and model systems whose set of control modes and switches is finite and, for each agent, given explicitly. Conversely, hybrid programs and hybrid process algebras offer infrastructures for defining modes and switches symbolically. Hybrid programs define modes and transitions implicitly using discrete variables, conditional choices, and possibly non-deterministic update rules. Hybrid process algebras define the discrete structure of the systems using events rather than variables, namely using process composition, non-deterministic choices, and communication and synchronization channels. While hybrid programs and hybrid process algebras offer the possibility to reason symbolically both about the discrete and the continuous component of the system, hybrid automata impose the discrete component to be explicit. Hybrid automata, while less compact, offer a simpler framework both for modeling and for reasoning about hybrid systems.

We focus our study on hybrid automata. Hybrid systems often find a natural description using hybrid automata. In fact, many safety critical time-based protocols and control systems, as we show in our experimental sections, often exhibit finite and simple discrete structure that interact against continuous dynamical systems with uncertainty. Hybrid automata are already challenging to analyze, even for the simplest verification question, that is reachability. Theoretically, the reachability analysis of hybrid automata is undecidable, in particular for the classes of hybrid systems we consider; in practice, it requires specific techniques for their analysis, in particular for the approximation of the continuous dynamics. In this work, we treat hybrid automata with (possibly) non-deterministic and (possibly) non-linear differential equations and inclusions, according to the following definition.

**Definition 1** (Hybrid automata)**.** A hybrid automaton $\mathcal{H}$ with $n$ real-valued variables consists of a finite directed multigraph $(V, E)$ where the vertices $v \in V$ are called control modes and the edges $e \in E$ are called control switches. Each $v \in V$ is decorated by an initial constraint $Z_v \subseteq \mathbb{R}^n$, denoting an initial condition of the form

$$x \in Z_v \tag{2.1}$$

over the variables $x$, an invariant constraint $I_v \subseteq \mathbb{R}^n$ and a flow constraint $F_v \colon \mathbb{R}^n \to \wp(\mathbb{R}^n)$, denoting a differential inclusion of the form

$$\dot{x} \in F_v(x),\ x \in I_v \tag{2.2}$$

over the variables $x$ and their derivatives $\dot{x}$. Each $e \in E$ is decorated by a jump constraint $J_e \colon \mathbb{R}^n \to \wp(\mathbb{R}^n)$, denoting a difference relation of the form

$$x' \in J_e(x), \tag{2.3}$$

over the variables $x$ and their successors $x'$.

The semantics of hybrid automata is given by the notions of control path and trajectory. A control path $v_0, e_1, v_1, \dots, e_k, v_k$ is a path of the control graph of $\mathcal{H}$, i.e., for all $0 \leq i \leq k$ it holds that $v_i \in V$ and for all $1 \leq i \leq k$ it holds that $e_i \in E$ and is a switch with source $v_{i-1}$ and destination $v_i$. When a control path is clear from the context, we abbreviate any object indexed by $v_i$ or $e_i$ as the same object indexed by $i$, e.g., we abbreviate flow $v_i$ as flow $i$. A state of the system is given by mode and a valuation for the variables, i.e., a pair $(v, x) \in V \times \mathbb{R}^n$. A trajectory is a possibly infinite sequence of states interleaved, in alternation, first by time elapses $t_0, t_1, \dots > 0$, and then by switches $e_0, e_1, \dots \in E$

$$(v_0, x_0)t_0(v_0, y_0)e_1(v_1, x_1)t_1(v_1, y_1)e_2 \dots, \tag{2.4}$$

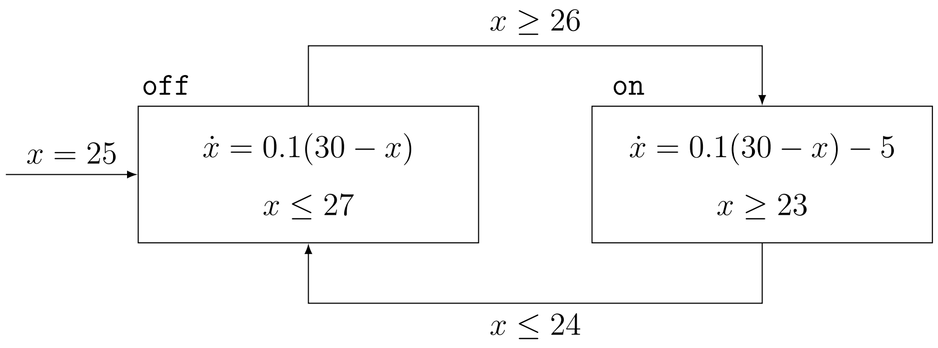


**Figure 2.2: Thermostat automaton of an air conditioner.**

for which there exists a sequence of solutions $\psi_0, \psi_1, \ldots : \mathbb{R} \to \mathbb{R}^n$ such that $\psi_i(0) = x_i$, $\psi_i(t_i) = y_i$, which, in their turn, satisfy (i) invariant constraints $\psi_i(t) \in I_i$ and (ii) the flow constraints $\dot{\psi}_i(t) \in F_i(\psi_i(t))$, for all $t \in [0, t_i]$. Moreover, $x_0 \in Z_0$ and, for every switch $e_i$, $v_{i+1}$ is a destination, $v_i$ is a source, and the respective states satisfy its jump condition, namely $x_{i+1} \in J_i(y_i)$. The semantics of a hybrid automaton is the maximal set of its trajectories. A hybrid automaton is safe if none of its trajectories contains a special bad mode.

Consider, for instance, the hybrid automaton in Fig. 2.2, which models the temperature controller of an air conditioner. Against an external temperature of 30 degrees, the system aims at keeping the internal temperature, the variable $x$, around 25 degrees. Initially, the internal temperature is 25 and the air conditioner is off. The thermostat switches the air conditioner on if $x$ rises above 26, and no later than when it reaches 27 degrees. Similarly, it switches off if $x$ falls below 24, no later then when it reaches 23. The automaton consists of polyhedral guards and invariants, and flows given by linear ordinary differential equations, which is the class of piece-wise affine hybrid automata (PWD). We tread the reachability analysis of PWD in Sec. 5 and Sec. 6. In this chapter, we limit our analysis to discrete-time systems, i.e., systems with derivative 0.

## 2.2 Template-polyhedral Abstraction

To illustrate, we consider hybrid automata where the jump condition of every switch $e \in E$ is characterized by a difference equation of the form

$$x' = A_e x + u, \ u \in U_e, \ x \in G_e, \tag{2.5}$$

where $A_e$ is a linear map, $U_e$ the input set, and $G_e$ the guard set, and the initial condition of each mode $v \in V$ by the set $Z_v$. In particular, we impose $Z_v$, $U_e$, and $G_e$ to be closed convex sets and $Z_v$ and $U_v$ to be bounded. While we treat properly continuous flows in Sec. 4 and 6, in this section we assume the flow to be constant, i.e., $\dot{x} = 0$ on each mode, in other words we treat (discrete time) linear time invariant (LTI) systems. For example,

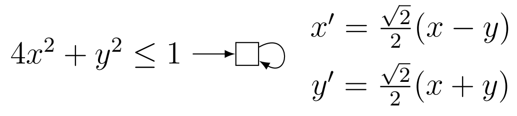

$4x^2 + y^2 \leq 1$
$x' = \frac{\sqrt{2}}{2}(x - y)$
$y' = \frac{\sqrt{2}}{2}(x + y)$

**Figure 2.3: A discrete time LTI system.**

we consider the 2-dimensional system depicted in Fig. 2.3, which repeatedly applies a counterclockwise $\pi/4$ rotation to an ellipse centered in the origin.

We abstract the reach set of the system by means of template polyhedra [103; 102]. Precisely, given a finite set of non-zero vectors $D = \{d_1, \dots, d_m\}$ in $\mathbb{R}^n$, which we call template directions, a *D-polyhedron* is a convex polyhedron in $\mathbb{R}^n$ for which every facet is normal to some vector in $D$. In other words, it is the set of solutions to the system of inequalities

$$\langle d_i, x \rangle \leq \delta_i, \ i = 1, \dots, m, \tag{2.6}$$

uniquely determined by the coefficients $\delta_1, \dots, \delta_m$, taken over $\mathbb{R} \cup \{+\infty\}$. Every finite coefficient denotes the signed distance of the respective facet from the origin in the respective direction, while every infinite coefficient denotes the absence of the facet. Notable examples of template polyhedra are the intervals and the octagons, from which template polyhedra inherit efficient inclusion testing. In fact, for every two $D$-polyhedra $P$ and $Q$ respectively represented by the coefficients $\delta_1, \dots, \delta_m$ and $\gamma_1, \dots, \gamma_m$ for $P \subseteq Q$ it is necessary and sufficient that $\delta_i \leq \gamma_i$ for all $i = 1, \dots, m$. On the other hand, they also inherit the wrapping effect which, in reachability analysis, may cause overapproximation errors which in their turn may lead into spurious counterexamples.

We associate a template $D_v$ to every mode $v \in V$ of the hybrid automaton, we explore its control graph, incrementally generating paths $v_0 e_1 v_1 e_2 v_2 \dots$ and computing sequences of template polyhedra $P_0, P_1, \dots$, each of which accounts for the reach set in correspondence of the respective mode along path. More precisely, we let $P_0$ be the tightest $D_0$-polyhedron enclosing $Z_0$ and $P_i$ be the tightest $D_i$-polyhedron enclosing $\text{jump}_i(P_{i-1})$ for every $i > 1$, where

$$\text{jump}_e(X) = A_e(X \cap G_e) + U_e. \tag{2.7}$$

The jump operator of Eq. 2.7 represents exactly all reached states after transition $e$ from every state in $X$. As a result, every template polyhedron overapproximates the reach set

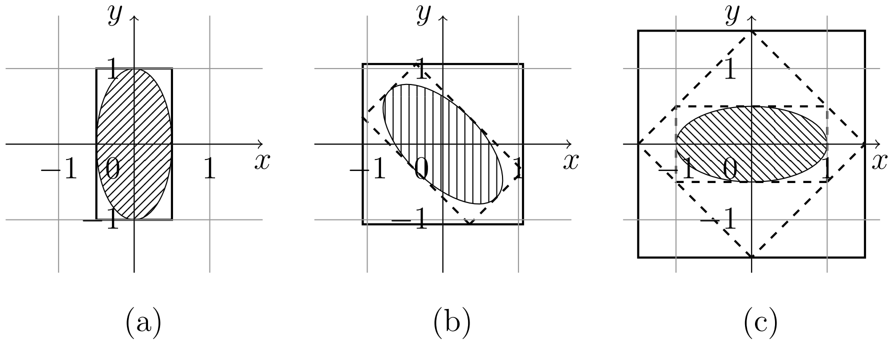


**Figure 2.4: A sequence of template polyhedral abstractions.**

of the corresponding mode as it happens, e.g., in Fig. 2.4 which shows the result of an abstraction of the system in Fig. 2.3. In particular, it depicts the first three polyhedra obtained using the interval template and also shows an example of wrapping effect: while the ellipse representing the exact react set rotates, the abstraction grows. The mitigation of the wrapping effect is our subject of study which, in Sec. 3.1, we tackle by template refinement, building on top of the idea of using support functions [59].

## 2.3 Support Functions

Support functions offer a framework to represent operations over convex sets precisely and efficiently, by (i) allowing to only compute tightest halfspaces in given directions and (ii) pushing the complexity to evaluation phase, in a lazy fashion. Formally, the *support function* of a convex set $X$ in $\mathbb{R}^n$ in a direction $d \in \mathbb{R}^n$ is

$$\rho_X(d) = \sup\{\langle d, x\rangle : x \in X\}, \tag{2.8}$$

which takes values from the extended real line $\mathbb{R} \cup \{+\infty, -\infty\}$ and is convex. Specifically, $\rho_X$ takes $+\infty$ if $X$ recedes in direction $d$, while takes $-\infty$ if $X$ is empty. A finite value indicate the position of the tightest halfspace including $X$, that is the *supporting halfspace*, whose outward pointing direction is $d$. For instance, for the ellipse $Z = \{(x, y) : 4x^2 + y^2 \leq 1\}$

depicted in Fig. 2.4a we have, e.g., that $\rho_Z(0,1) = 1$ and $\rho_Z(1,0) = 0.5$. Consequently, the intersection of respective supporting halfspaces defines exactly the tightest $D$-polyhedron enclosing $X$, which is given by the system

$$\langle d, x \rangle \leq \rho_X(d), \; d \in D. \tag{2.9}$$

Thus, to reason about template polyhedral abstractions one can simply reason about the support function of the reach set or, more precisely, about their properties with respect to set operations induced by the post operators.

Reachability analysis produces a collection of polyhedra that are related with one another by post operators, in the current case by the operator jump. The operator jump acts as a set transformer, which applies a Minkowski sum to a linear transformation to an intersection; dually, the same operations can be applied to their support functions. More concretely, the support function of the Minkowski sum of two non-empty convex sets $X$ and $Y$ in $\mathbb{R}^n$ and the support function of a linear transformation $A$ from a convex set $X$ in $\mathbb{R}^n$ to $\mathbb{R}^m$ respectively amount to

$$\rho_{X+Y}(d) = \rho_X(d) + \rho_Y(d), \tag{2.10}$$

$$\rho_{AX}(d) = \rho_X(A^\mathsf{T} d). \tag{2.11}$$

As a result, we can compute arbitrary nestings of Minkowski sums and linear transformations from a set of known support functions, first computing their argument directions and then composing the resulting values. Intersection can be treated similarly, for that

$$\rho_{X \cap Y}(d) \leq \min\{\rho_X(d), \rho_Y(d)\}; \tag{2.12}$$

unfortunately, this may introduce overapproximation as, e.g., it occurs to any two non-empty but disjoint sets. Reasoning about disjointedness and, more generally, about precise intersections is key in template refinement but requires the online search of appropriate argument directions.

The exact support function of an intersection is the closure of the infimal convolution of their support functions [99, Cor. 16.4.1]. More formally, the *infimal convolution* of a set of convex functions $f_1, \dots, f_m$ is

$$f_1 * \cdots * f_m(x) = \inf\{f_1(x'_1) + \cdots + f_m(x'_m) \colon x'_1 + \cdots + x'_m = x\}. \tag{2.13}$$

The *closure* of a convex function $f$, denoted $\operatorname{cl} f$, is the greatest lower semi-continuous function majorized by $f$, if $f(x) > -\infty$ for all $x$; it is the constant function $-\infty$, otherwise. As it turns out, for every $X_1, \dots, X_m$ that are non-empty closed convex sets in $\mathbb{R}^n$ the support function of their intersection amounts to

$$\rho_{X_1 \cap \dots \cap X_m} = \operatorname{cl}(\rho_{X_1} * \dots * \rho_{X_m}). \tag{2.14}$$

However, under certain circumstances the closure can be dropped. For instance, this is the case when all relative interiors intersect [99, Cor. 16.4.1] and when all sets intersect and are polyhedral [99, Thm. 20.1]; above all, when all sets do not recede in the same direction, whether they intersect or not.

**Theorem 2.3.1.** *If $X_1, \dots, X_m$ are non-empty closed convex sets in $\mathbb{R}^n$ that have no common direction of recession then $\rho_{X_1} * \dots * \rho_{X_m}$ is closed, hence*

$$\rho_{X_1 \cap \dots \cap X_m} = \rho_{X_1} * \dots * \rho_{X_m}. \tag{2.15}$$

*Proof.* By the recession condition, we have that for every direction $d$ at least one support function is bounded above, i.e., w.l.o.g. $\rho_{X_1}(d) < +\infty$. Then, from the infimal convolution formula in Eq. 2.13 we have (by setting $x_2 = \dots = x_m = 0$) $\rho_{X_1} * \dots * \rho_{X_m}(d) \leq \rho_{X_1}(d) + \rho_{X_2}(0) + \dots + \rho_{X_m}(0) = \rho_{X_1}(d) < +\infty$. Hence the infimal convolution is bounded above everywhere, in other words

$$\operatorname{dom}(\rho_{X_1} * \dots * \rho_{X_m}) = \mathbb{R}^n. \tag{2.16}$$

If $\rho_{X_1} * \dots * \rho_{X_m}$ turns out to be proper, i.e., bounded below, we know its closure can differ only on relative boundary points of the effective domain [99, Thm. 7.4]. But $\mathbb{R}^n$ has no boundary points at all, hence $\rho_{X_1} * \dots * \rho_{X_m}$ is closed. If it turns out to be improper, i.e., $-\infty$ at some point, then we know it must be $-\infty$ at every point of the relative interior of its effective domain [99, Thm. 7.2]. But $\operatorname{ri} \mathbb{R}^n = \mathbb{R}^n$, hence $\rho_{X_1} * \dots * \rho_{X_m}$ equals $-\infty$ everywhere. $\square$

Support functions offer a framework for composing set operations efficiently, and delay the complexity to the moment of computing the supporting halfspaces for a template polyhedron. Moreover, support function pose the basic theory for the refinement of template directions, which we introduce in Ch. 3.

# 3 Automatic Template Refinement

The counterexample-guided abstraction refinement for reachability analysis using template polyhedra consist of the interaction of an abstractor, which either declares the system safe or returns a (possibly spurious) counterexample, and a refiner, which either declares the counterexample feasible or returns a template which eliminates it. The abstractor constructs a polyhedral enclosure for the reach set, until it finds a fixpoint or a abstract counterexample, associated with an initial set $X$ in $\mathbb{R}^n$, a finite sequence of post operators $f_1, \ldots, f_k$ from set in $\mathbb{R}^n$ to set in $\mathbb{R}^n$, each of which corresponds to the semantics of the respective step, and a finite sequence of template polyhedra $P_1, \ldots, P_k$ in $\mathbb{R}^n$, each of which complies to the respective template among $D_1, \ldots, D_k$. The abstract path is a candidate counterexample, in the sense that the given polyhedra make it feasible; in other words,

$$X \subseteq P_1, f_1(P_1) \subseteq P_2, f_2(P_2) \subseteq P_3, \ldots, f_{k-1}(P_{k-1}) \subseteq P_k, f_k(P_k) \neq \emptyset. \tag{3.1}$$

The refiner decides whether the counterexample is actually genuine, namely it decides whether $f_k \circ \cdots \circ f_1(X) \neq \emptyset$. Otherwise, if spurious, the refiner computes a sequence of templates $D'_1, \ldots, D'_k$ that refine the previous ones, i.e., $D'_1 \supseteq D_1, \ldots, D'_k \supseteq D_k$, and such that the tightest and respectively complying template polyhedra $P'_1, \ldots, P'_k$ satisfy

$$X \subseteq P'_1, f_1(P'_1) \subseteq P'_2, f_2(P'_2) \subseteq P'_3, \ldots, f_{k-1}(P'_{k-1}) \subseteq P'_k, f_k(P'_k) = \emptyset. \tag{3.2}$$

As a result, the successive abstraction will necessarily report a different counterexample.

Template refinement is a form Craig's interpolation. More concretely, for a $X$ in $\mathbb{R}^n$, a transformation $f$ from set in $\mathbb{R}^n$ to set in $\mathbb{R}^n$, and a set $Y$ in $\mathbb{R}^n$, we consider the interpolant for the inclusion $X \subseteq f^{-1}(Y)$, which consists of some set $I$ in $\mathbb{R}^n$ such that

$$X \subseteq I \text{ and } I \subseteq f^{-1}(Y) \tag{3.3}$$

or, in other words, such that $X \subseteq I$ and $f(I) \subseteq Y$. Hence, indeed, the polyhedra in Eq. 3.2 can be seen as an inductive sequence of interpolants for $X$, the transformations $f_1, \dots, f_k$, and the empty set; in fact, with $X_i = f_i \circ \cdots \circ f_1(X)$, we have that $P'_k$ interpolates the inclusion $f_k(X_{k-1}) \subseteq \emptyset$, $P'_{k-1}$ interpolates $f_{k-1}(X_{k-2}) \subseteq P'_k$, and so on. We observe that, if we can compute any such sequence of polyhedral interpolants, then the outwards pointing directions of the halfspaces defining the polyhedra also define valid refining templates. Indeed, it is sufficient to compute a sequence of halfspaces $H'_1, \dots, H'_k$ in $\mathbb{R}^n$ (possibly including $\emptyset$ and $\mathbb{R}^n$) such that

$$X \subseteq H'_1, f_1(H'_1) \subseteq H'_2, f_2(H'_2) \subseteq H'_3, \dots, f_{k-1}(H'_{k-1}) \subseteq H'_k, f_k(H'_k) = \emptyset. \tag{3.4}$$

We take the respective outward pointing directions $a_1, \dots, a_k \in \mathbb{R}^n$ and add them to the templates $D_1, \dots D_k$, obtaining $D'_i = D_i \cup \{a_i\}$, for $i = 1, \dots, k$. As a consequence, the

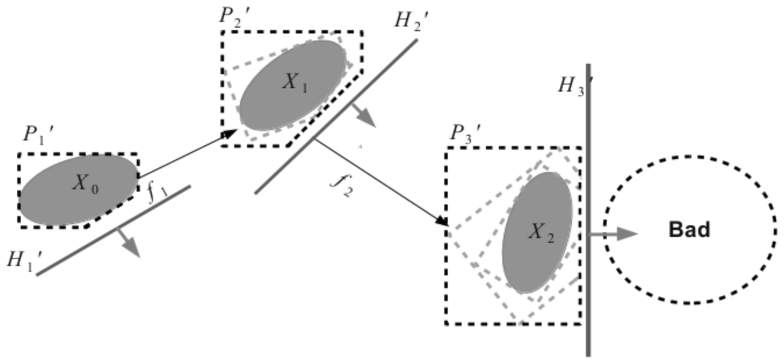


**Figure 3.1: Template refinement using halfspace interpolants.**

tightest $D'_i$-polyhedron $P'_i$ must be necessarily included in $P_i \cup H'_i$, hence, refining the abstraction and eliminating the path, as we illustrate in Fig. 3.1.

Halfspace interpolants are sufficient for refinement; next, we wonder whether they are necessary. First, it is necessary to interpolate at every step along the path. Consider, e.g., Fig. 3.1, where operators $f_1$ and $f_2$ apply shift and rotation, $f_3$ makes an intersection and causes infeasibility of the path: it is not sufficient to refine $D_3$ only [], but necessary to add directions to $D_1$ and $D_2$, too. Second, for infeasible paths interpolants exist and are necessarily halfspaces, under conditions over sets and operators like, e.g., linearity or compactness, as we discuss in Sec. 3.1, and are computable using convex optimizations, as we discuss in Sec. 3.2.

## 3.1 Halfspace Interpolation

We establish the existence of a sequence of halfspace interpolants by first studying whether each operator along the path admits a halfspace containing its operator provided a halfspace containing its result. We investigate the halfspace interpolation for the operators of Minkowski sum, linear map, and intersection of convex sets that are closed. In particular, while for the first two operators halfspace interpolation seems to work, for the intersection operator it may fail even with simple examples. Consider the unary operator $f(X) = X \cap Y$ that takes a closed convex set $X$ and returns its intersection with the closed convex set $Y$. Provided a halfspace $H$ that contains $X \cap Y$, we ask whether there exists a (proper) halfspace interpolant, namely a halfspace $H'$ that contains $X$ and for which $H' \cap Y \subseteq H$, in other words containing $X$ and disjoint from $Y \backslash H$. Unfortunately, this is not the always the case. Consider the example in Fig. 3.2, where $X$ is the unit ball centered in $(-1, 0)$,

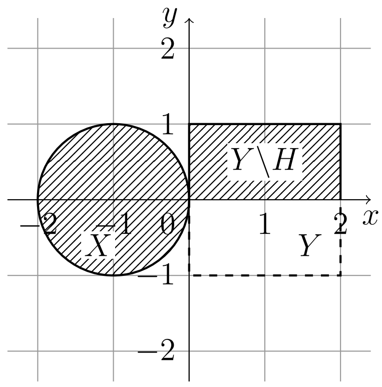


**Figure 3.2: Intersection that does not allow proper halfspace interpolation.**

$Y$ be the unit box centered in $(1, 0)$, and $H$ is $\{(x, y)\colon y \leq 0\}$. The only halfspace that contains $X$ and is disjoint with the interior of $Y \backslash H$ is $\{(x, y)\colon x \leq 0\}$, which unfortunately intersects the boundary of $Y \backslash H$. On the contrary, if we move $H$ as to impose a gap between $Y \backslash H$ and $X \cap Y$, the origin, then a halfspace interpolant exists. We generalize the idea of imposing a gap around halfspace interpolants with the notion strong halfspace interpolation and we prove that strong halfspace interpolants exist under conditions for the constraints such as, e.g., boundedness or linearity.

Strong halfspace interpolants are halfspace interpolants that strongly include the respective operands. Precisely, a set $Y$ includes a set $X$ strongly if there exists some $\varepsilon > 0$ such that $X + \varepsilon B \subseteq Y$, where $B$ is the unit Euclidean ball $\{x \colon |x| \leq 1\}$; for known $\varepsilon$, we denote strong inclusion as $X \subseteq_\varepsilon Y$. Consequently, an $m$-ary operator $f$ admits strong halfspace interpolation if for every set of operands $X_1, \dots, X_m$ and halfspace $H$ that includes $f(X_1, \dots, X_m)$ strongly there exists some halfspaces $H'_1, \dots, H'_m$ that respectively include $X_1, \dots, X_m$ strongly and for which $H$ includes $f(H'_1, \dots, H'_m)$ strongly. When $X_1, \dots, X_m$, $f$, and $H$ are clear from the context we call such $H'_1, \dots, H'_m$ strong interpolants. Notably, strong interpolation first assumes the result the operation to be strongly included and then guarantees the operands to be strongly included. As a result, for any arbitrarily long sequence of operations, each of which admits strong halfspace interpolation, we can inductively construct a sequence of strong halfspace interpolants to prove, e.g., emptiness of the result. To this aim, we show that the operations of intersection (bounded or linear), Minkowski sum, and linear transformation admit strong halfspace interpolation, and that strong halfspace interpolation for operations extends to sequences of operations.

The operation of intersection emerges from the abstraction of the semantics of guards and invariant of a hybrid automaton, which in their turn may emerge from intersections of multiple constraints. The strong halfspace interpolation of arbitrary intersections of closed convex sets holds, under the sufficient condition that the infimal convolution of their support functions is closed.

**Theorem 3.1.1.** *Let $X_1, \dots, X_m$ be closed convex sets in $\mathbb{R}^n$. For the intersection of $X_1, \dots, X_m$ to admit strong halfspace interpolation it is sufficient that $\rho_{X_1} * \cdots * \rho_{X_m}$ is closed.*

*Proof.* Let $H = \{x \colon \langle a, x\rangle \leq b\}$ be a halfspace such that $X_1 \cap \cdots \cap X_m \subseteq_\varepsilon H$ for some $\varepsilon > 0$. We first assume that $X_1, \dots, X_m$ are non-empty and observe that, by Eq. 2.14 and the closeness hypothesis [99, Cor. 16.4.1], we have that $X_1 \cap \cdots \cap X_m \subseteq_\varepsilon H$ translates to

$$\inf\{\rho_{X_1}(a'_1) + \cdots + \rho_{X_m}(a'_m) \colon a'_1 + \cdots + a'_m = a\} + \varepsilon|a| \leq b. \tag{3.5}$$

Under the assumption that $H \neq \mathbb{R}^n$, there exists some $c \in \mathbb{R}$ such that

$$\inf\{\rho_{X_1}(a'_1) + \cdots + \rho_{X_m}(a'_m) \colon a'_1 + \cdots + a'_m = a\} < c < b. \tag{3.6}$$

In fact, if $H \neq \emptyset$ then $\varepsilon|a| > 0$ and, therefore, the existence of $c$ follows from Eq. 3.5. If otherwise $H = \emptyset$ then $X_1 \cap \cdots \cap X_m = \emptyset$ and, therefore, the infimum in Eq. 3.6 equals $-\infty$ and $c$ trivially exists. Altogether, since $c$ is strictly greater than the infimum, there must exist (by definition of infimum) $a'_1, \ldots, a'_m$ such that $a'_1 + \cdots + a'_m = a$ and $\rho_{X_1}(a'_1) + \cdots + \rho_{X_m}(a'_m) \leq c$, which also satisfy

$$\rho_{X_1}(a'_1) + \cdots + \rho_{X_m}(a'_m) + (b - c) \leq b. \tag{3.7}$$

Since $c < b$, there also exists some $\varepsilon' > 0$ such that $\varepsilon'(|a| + \sum_{i=1}^{m} |a'_i|) \leq b - c$. Then, by substituting $(b - c)$ with $\varepsilon'(|a| + \sum_{i=1}^{m} |a'_i|)$ in Eq. 3.7, we obtain

$$\rho_{X_1}(a'_1) + \varepsilon'|a'_1| + \cdots + \rho_{X_m}(a'_m) + \varepsilon'|a'_1| + \varepsilon'|a| \leq b. \tag{3.8}$$

We let $b'_i = \rho_{X_i}(a'_i) + \varepsilon'|a'_i|$ and $H_i = \{x \colon \langle a'_i, x\rangle \leq b'_i\}$ and show that $H_1, \ldots, H_m$ constitute strong interpolants. First, for every halfspace we have that $X_i \subseteq_{\varepsilon'} H_i$ which, in fact, is equivalent to saying $\rho_{X_i}(a'_i) + \varepsilon'|a'_i| \leq b'_i$. Second, (by definition of closure) we have that $\rho_{H'_1 \cap \cdots \cap H'_m}(a) \leq \rho_{H'_1} * \cdots * \rho_{H'_m}(a)$ and (by definition of infimum) that $\rho_{H'_1} * \cdots * \rho_{H'_m}(a) \leq \rho_{H'_1}(a'_1) + \cdots + \rho_{H_m}(a'_m)$, for every $a'_1 + \cdots + a'_m = a$. Since we know that $\rho_{H'_1}(a'_1) = b'_i$ then, by Eq. 3.8, we have

$$\rho_{H'_1 \cap \cdots \cap H'_m}(a) + \varepsilon'|a| \leq b'_1 + \cdots + b'_m + \varepsilon'|a| \leq b, \tag{3.9}$$

in other words $H'_1 \cap \cdots \cap H'_m \subseteq_{\varepsilon'} H$, which concludes the proof under the assumption that $X_1, \ldots, X_m$ are non-empty and $H \neq \mathbb{R}^n$. Finally, if either set is empty, say w.l.o.g that $X_1 = \emptyset$, we simply take $H'_1 = \emptyset$ and $H'_2 = \cdots = H'_m = \mathbb{R}^n$. If $H = \mathbb{R}^n$ we simply take $H'_1 = \cdots = H'_m = \mathbb{R}^n$. In both cases, $H'_1, \ldots, H'_m$ constitute strong interpolants for any value of $\varepsilon'$. □

Closedness applies for intersection forming bounded sets; more generally, for intersections of sets without common direction of recession.

**Corollary 3.1.1.1.** *The intersection of a finite number of closed convex sets in $\mathbb{R}^n$ without common direction of recession admits strong halfspace interpolation.*

*Proof.* It follows from Thm. 2.3.1 and 3.1.1. □

The intersection of Fig. 3.2 allows strong halfspace interpolation as, in fact, it satisfies the recession condition. In particular, both $X$ and $Y$ have no direction of recession at all, as a consequence of being bounded; more generally, the recession condition holds when either set is bounded.

**Corollary 3.1.1.2.** *The intersection of a finite number of closed convex sets in $\mathbb{R}^n$, where at least one is bounded, admits strong halfspace interpolation.*

*Proof.* Cor. 3.1.1.1 applies, as a bounded set have no directions of recession at all. $\square$

Strong halfspace interpolation generalizes strong halfplane separation in the sense that, for $X \cap Y$ strongly included in the empty halfspace $H = \emptyset$ and, hence, $X$ and $Y$ disjoint, strong halfspace interpolants exist when $X$ and $Y$ are strongly separated; in other words, when there exists a halfplane whose corresponding two halfspaces respectively include $X$ and $Y$, strongly. Consequently, disjoint closed convex sets that do not admit strong separation may not admit strong halfspace interpolation, which occurs to certain (non-linear) sets that do not satisfy the recession condition. For instance, take the sets $X = \{(x, y) \colon y \geq 1/x, x \geq 0\}$

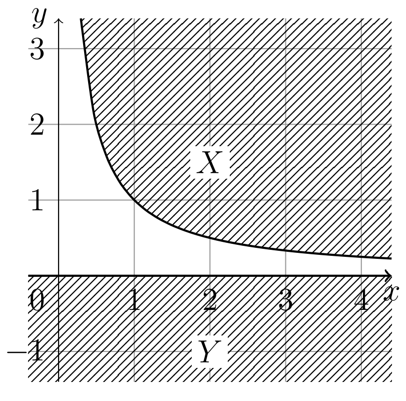


**Figure 3.3: Intersection that does not allow strong halfspace interpolation.**

and $Y = \{(x, y) \colon y \leq 0\}$, depicted in Fig. 3.3: no halfspace can strongly include $X$ without intersecting $Y$. On the contrary, strong halfspace interpolation always succeeds if all sets are polyhedral, without any further restriction.

**Theorem 3.1.2.** *The intersection of a finite number of polyhedra in $\mathbb{R}^n$ admits strong halfspace interpolation.*

*Proof.* Let $X_1 = \{x \colon A_1 x \leq b_1\}$, …, $X_m = \{x \colon A_m x \leq b_m\}$ be polyhedra in $\mathbb{R}^n$. If $X_1 \cap \cdots \cap X_m \neq \emptyset$ then $\rho_{X_1} * \cdots * \rho_{X_m}$ is closed [99, Thm. 20.1] and, by Thm. 3.1.1, the statement follows. If $X_1 \cap \cdots \cap X_m = \emptyset$ then, by Farkas' lemma, we have that

$$A_1^{\mathsf{T}} \lambda_1 + \cdots + A_m^{\mathsf{T}} \lambda_m = 0,\ \langle b_1, \lambda_1 \rangle + \cdots + \langle b_m, \lambda_m \rangle < 0, \tag{3.10}$$

for some vectors $\lambda_1, \ldots, \lambda_m \geq 0$, i.e., where all elements of the vectors are non-negative. Let $\varepsilon = 0 - \langle b_1, \lambda_1 \rangle + \cdots + \langle b_m, \lambda_m \rangle$, then $\varepsilon > 0$. Let $\varepsilon' = \varepsilon/(m+1)$ and $a_i' = A_i^{\mathsf{T}} \lambda_i$ and $b_i' = \langle b_i, \lambda_i \rangle + \varepsilon'$, for $i = 1, \ldots, m$. Then, we show that the halfspaces $H_i' = \{x \colon \langle a_i', x \rangle \leq b_i'\}$, for $i = 1, \ldots, m$, are valid strong halfspace interpolants. First, we have that

$$A_i^T \lambda = a_i',\ \langle b_i, \lambda_i \rangle + \varepsilon' \leq b_i' \tag{3.11}$$

which, by (weak) duality of linear programming, implies $\rho_{X_i}(a_i') + \varepsilon' \leq b_i'$ which, in its turn, implies $X_1$ strongly included in $H_i'$. Second, we have that

$$a_1' + \cdots + a_m' = 0,\ b_1' + \cdots + b_m' < 0 \tag{3.12}$$

which, by Farkas' lemma, implies that $H_1' \cap \cdots \cap H_m' = \emptyset$ which, trivially, is strongly included in any halfspace. □

The operators of Minkowski sum and linear transformation, unlike intersection, do not impose restrictions to the operands other than convexity.

**Theorem 3.1.3.** *Every Minkowski sum of two convex sets in $\mathbb{R}^n$ admits strong halfspace interpolation.*

*Proof.* Let $X$ and $Y$ be closed convex sets in $\mathbb{R}^n$ and $H = \{x \colon \langle a, x \rangle \leq b\}$ be a halfspace such that $X + Y \subseteq_\varepsilon H$, for some $\varepsilon > 0$. If $X$ and $Y$ are non-empty, then $\rho_{X+Y}(a) = \rho_X(a) + \rho_Y(a)$ and the hypothesis of strong inclusion translates into

$$\rho_X(a) + \rho_Y(a) + |a|\varepsilon \leq b. \tag{3.13}$$

Let $\varepsilon' = \varepsilon/3$. If $a \neq 0$ then $H \neq \mathbb{R}^n$ and, therefore, $\rho_X(a)$ and $\rho_Y(a)$ are finite values. We let $H' = \{x \colon \langle a, x \rangle \leq \rho_X(a) + |a|\varepsilon'\}$, $H'' = \{x \colon \langle a, x \rangle \leq \rho_Y(a) + |a|\varepsilon'\}$, and show they are strong halfspace interpolants. First, we have that $X \subseteq_{\varepsilon'} H'$ and $Y \subseteq_{\varepsilon'} H''$, by definition of $H'$ and $H''$. Second, we have that $H' + H'' \subseteq_{\varepsilon'} H$; in fact, $\rho_{H'+H''+\varepsilon' B}(a) = \rho_{H'}(a) + \rho_{H''}(a) + |a|\varepsilon' = \rho_X(a) + |a|\varepsilon' + \rho_Y(a) + |a|\varepsilon' + |a|\varepsilon'$ and hence, by Eq. 3.13, we have $\rho_{H'+H''+\varepsilon' B}(a) \leq b$, which concludes the proof for $a \neq 0$. If $a = 0$ then $b \geq 0$, because $\emptyset \neq X + Y \subseteq H$; consequently, $H = \mathbb{R}^n$ and $H' = H'' = \mathbb{R}^n$ are valid strong halfspace interpolants. Finally, if either $X$ or $Y$ are empty, take $H' = \emptyset$ for the empty one and $H'' = \mathbb{R}^n$ for the other; then $H' + H'' = \emptyset$, trivially included by $H$ strongly. □

**Theorem 3.1.4.** *Every linear transformation of a convex set in $\mathbb{R}^n$ to $\mathbb{R}^m$ admits strong halfspace interpolation.*

*Proof.* Let $X$ be a convex set in $\mathbb{R}^n$, $A$ a linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^m$, and $H = \{x\colon \langle a, x\rangle \leq b\}$ that contains $AX$ strongly, i.e., $AX \subseteq_\varepsilon H$ for some $\epsilon > 0$. First, let us assume $\emptyset \neq X \neq \mathbb{R}^n$. Since $\rho_{AX}(a) = \rho X(A^T a)$, we have that

$$\rho_X(A^T a) + |a|\varepsilon \leq b. \tag{3.14}$$

If $A^\mathsf{T} a \neq 0$, then we define $\varepsilon' = \frac{\varepsilon|a|}{2|A^\mathsf{T} a|}$ and $H' = \{x\colon \langle A^\mathsf{T} a, x\rangle \leq b - \varepsilon'|A^\mathsf{T} a|\}$, and rephrase Eq. 3.14 as

$$\rho_X(A^T a) + \varepsilon'|A^\mathsf{T} a| \leq b - \varepsilon'|A^\mathsf{T} a|. \tag{3.15}$$

First, by Eq. 3.15, we have that $X \subseteq_{\varepsilon'} H'$. Next, we define $\varepsilon'' = \frac{\varepsilon'|A^\mathsf{T} a|}{|a|}$ and $H'' = \{x\colon \langle a, x\rangle \leq b - \varepsilon''|a|\}$. First, we have $\rho_{AH'}(a) = b - \varepsilon'|A^\mathsf{T} a| = b - \varepsilon''|a| = \rho_{H''}(a)$, namely, $AH' = H''$; second, we also have $\rho_{H''}(a) + \varepsilon''|a| = b - \varepsilon''|a| + \varepsilon''|a| = b$, namely, $H'' + \varepsilon'' B = H'$. As a result, $X \subseteq_{\varepsilon'} H'$ and $AH' \subseteq_{\varepsilon''} H$; in other words, $H'$ constitutes a strong interpolant. Instead, if $A^\mathsf{T} a = 0$, we take any halfspace $H' \neq \mathbb{R}^n$ that includes $X$ strongly. Since $X \neq \emptyset$ and $A^\mathsf{T} a = 0$, we have $\rho_X(A^\mathsf{T} a) = 0$; hence, by Eq. 3.14, we have $|a|\varepsilon \leq b$. As a result, $\rho_{AH'}(a) + |a|\varepsilon = \rho_{H'}(A^\mathsf{T} a) + |a|\varepsilon = |a|\varepsilon \leq b$; in other words, $AH' \subseteq_\varepsilon H$. Finally, if $X = \emptyset$, we simply take $H' = \emptyset$; if $X = \mathbb{R}^n$, we simply take $H' = \mathbb{R}^n$. □

Notably, both Minkowski sum and linear transformation do not impose closedness of the operands. On the other hand, intersection, on top of the recession condition or linearity, requires closedness of the operands. For this reason, when we have a composition of multiple operators (involving intersections), we need to ensure that all operands preserve closedness. Unfortunately, closedness is not preserved, in general, by Minkowski sums and linear transformations. For instance, the closed set $X = \{(x, y)\colon y \geq 1/x, x \geq 0\}$ in Fig. 3.3 through the projection $A(x, y) = x$ is open; conversely, closedness is preserved if, e.g., the operand is compact or the transformation is invertible [99, Thm. 9.1]. Another example is the Minkowski sum of $X = \{(x, y) \colon y \geq 1/x, x \geq 0\}$ and $Y = \{(x, y) \colon y \geq 0\}$, which is the open set $X + Y = \{(x, y) \colon y > 0)\}$. Conversely, for sets, e.g., whose directions of recession are never opposite to each other, the Minkowski sum is closed [99, Cor. 9.1.2].

The composition of a sequence of unary operators, for which every output complies with the input of the subsequent operator, admits sequences of strong halfspace interpolants if every operator admits strong interpolation.

**Theorem 3.1.5.** *Let $f_1, \dots, f_k$ be a sequence of unary operators that admit strong halfspace interpolation. For every operand $X$ and halfspace $H$ that includes $f_k \circ \cdots \circ f_1(X)$ strongly there exists a sequence of halfspaces $H'_1, \dots, H'_k$ such that*

$$X \subseteq_\varepsilon H'_1, f_1(H'_1) \subseteq_\varepsilon H'_2, \dots, f_{k-1}(H'_{k-1}) \subseteq_\varepsilon H'_k, f_k(H'_k) \subseteq_\varepsilon H \qquad (3.16)$$

*for some $\varepsilon > 0$.*

*Proof (sketch).* Let $X_1, \dots X_k$ be the sets given by $f_1(X) = X_1, f_2(X_1) = X_2, \dots, f_k(X_{k-1}) = X_k$. By induction, backward, we have that, if there exists $\varepsilon_i > 0$ and a halfspace $H_i$ such that $X_i \subseteq_{\varepsilon_i} H'_i$, then there exists $\varepsilon_{i-1} > 0$ and halfspace $H_{i-1}$ such that $X_{i-1} \subseteq_{\varepsilon_{i-1}} H'_{i-1}$. As for the base case, we have that $X_k$ is strongly included in $H$, by hypothesis. As for $\varepsilon > 0$, take the minimum between $\varepsilon_1, \dots, \varepsilon_k$. □

One should note that unary operators can be constructed from generic $m$-ary operators, by simply fixing all but one arguments. Also, for unary operators, strong halfspace interpolation is preserved by composition.

**Corollary 3.1.5.1.** *If $f$ and $g$ admit strong halfspace interpolation then $f \circ g$ admits strong halfspace interpolation.*

*Proof.* Apply Thm. 3.1.5 for $k = 2$. □

Besides, interpolation upon emptiness of Eq. 3.4 is a special case of interpolation upon strong inclusion and, hence, is equally satisfied. Additionally, upon emptiness there must exist a minimal sequence of non-empty interpolants, from the initial input up to the very operator causing the result to be empty.

**Corollary 3.1.5.2.** *Let $f_1, \dots, f_k$ be a sequence of monotonic unary operators that admit strong halfspace interpolation and $X$ an operand such that $f_k \circ \cdots \circ f_1(X)$ is empty. For some $1 \leq j \leq k$ there exists a sequence of non-empty halfspaces $H'_1, \dots, H'_j$ such that*

$$X \subseteq_\varepsilon H'_1, f_1(H'_1) \subseteq_\varepsilon H'_2, \dots, f_{j-1}(H'_{j-1}) \subseteq_\varepsilon H'_j, f_j(H'_j) = f_{j+1}(\emptyset) = \cdots = f_k(\emptyset) = \emptyset, \tag{3.17}$$

*for some $\varepsilon > 0$.*

*Proof (sketch).* Apply Thm. 3.1.5 to the minimal subsequence $f_1, \dots, f_j$ for which inclusion within $H = \emptyset$ holds. Since $f_k \circ \cdots \circ f_1(X)$ is empty, such $j$ must exist. Moreover, $H_1, \dots, H_j$ must be non-empty, otherwise they would contradict minimality. Also, the extension with $H_{j+1} = \cdots = H_k = \emptyset$ constitutes a valid sequence of interpolants for $f_1, \dots, f_k$, by monotonicity of the operators. □

We have established that, for sequences of operators involving Minkowski sum, linear transformations, and intersections of linear or convex sets without common direction of recession, sequences of halfspace interpolants always exist. Besides, for operators over (convex) semi-algebraic sets, halfspace interpolants are computable but, using general procedures such as the cylindrical algebraic decomposition, cab be very expensive. Nevertheless, as we show in Sec. 3.2, halfspace interpolation can be translated into convex programming for which, according to the kind of sets, efficient procedures are available.

## 3.2 Interpolation as Convex Optimization

Convex optimization comprises several methods for computing extrema of convex functions within convex constraints, whose decision fragment consists of deciding the existence of a point within the convex constraints. We translate the halfspace interpolation problem for arbitrary compositions of Minkowski sums, linear transformations, and intersections into the feasibility problem for convex constraints, amenable to modern convex optimizers. We also show that, an optimization variant of the same problem, corresponds to the support function of the result of the same composition.

We consider $m$-ary intersections, Minsk's sums, and linear transformations, whose arguments are either the results of other $m$-ary intersections, Minsk's sums, or linear transformations, or sets defined by constraints. In particular, to the result of every operation and to every set defined by constraints, both of which we generically call $X$, we associate a variable $a_X$ that takes from $\mathbb{R}^m$, where $m$ is the dimensionality of $X$, and a variable $b_X$ that takes from $\mathbb{R}$. Then, we construct our encoding over the structure of the composition, according to the rules

$$a_{X_1} + \cdots + a_{X_m} = a_{X_1 \cap \cdots \cap X_m} \qquad b_{X_1} + \cdots + b_{X_m} \le b_{X_1 \cap \cdots \cap X_m}, \tag{3.18}$$

$$a_X = a_Y = a_{X+Y} \qquad b_X + b_Y \le b_{X+Y}, \tag{3.19}$$

$$a_X = A^{\mathsf{T}} a_{AX} \qquad b_X \le b_{AX}. \tag{3.20}$$

Interpolation with respect to inclusion of the whole composition, which we call $X$, within the halfspace $H = \{x \colon \langle a_H, x\rangle \le b_H\}$, consists of imposing the additional constraint

$$b_X = a_H \qquad b_X \le b_H. \tag{3.21}$$

To make an example, consider a single application of the discrete post operator of a LTI system, shown in Eq. 2.7, that is the set $A(X \cap G) + U$; then, inclusion with respect to $H$ gives the encoding

$$\begin{aligned} a_{A(X\cap G)+U} &= a_H & b_{A(X\cap G)+U} &\le b_H, \\ a_{A(X\cap G)} = a_U &= a_{A(X\cap G)+U}, & b_{A(X\cap G)} + b_U &\le b_{A(X\cap G)+U}, \\ a_{X\cap G} &= A^{\mathsf{T}} a_{A(X\cap G)} & b_{X\cap G} &\le b_{A(X\cap G)}, \\ a_X + a_G &= a_{X\cap G} & b_X + b_G &\le b_{X\cap G}. \end{aligned} \tag{3.22}$$

It remains to encode the bottom sets, i.e., those defined by explicit constraints; in the example, these are $X$, $G$, and $U$. The encoding for the bottom sets is constructed using the respective theories of duality, which we discuss later. In the meanwhile, independently of the bottom constraints, we show that a solution to our encoding constitutes halfspace interpolants. Precisely, for a variable assignment satisfying the system of constraint, the halfspace interpolant respectively including each set $X$ is given by

$$H_X = \{x \colon \langle a_X, x\rangle \le b_X\}. \tag{3.23}$$

Concerning the example in Eq. 3.22, the respective interpolants are given by

$$\begin{aligned} H_{A(X\cap G)+U} = \{x\colon \langle a_{A(X\cap G)+U}, x\rangle \leq b_{A(X\cap G)+U}\},\ H_U = \{x\colon \langle a_U, x\rangle \leq b_U\}, \\ H_{A(X\cap G)} = \{x\colon \langle a_{A(X\cap G)}, x\rangle \leq b_{A(X\cap G)}\},\ H_G = \{x\colon \langle a_G, x\rangle \leq b_G\}, \\ H_{X\cap G} = \{x\colon \langle a_{X\cap G}, x\rangle \leq b_{X\cap G}\},\ H_X = \{x\colon \langle a_X, x\rangle \leq b_X\}. \end{aligned} \tag{3.24}$$

To show the correctness of the encoding, we show that it is sound, in the sense that every solution constitutes interpolants; then, we show that it is also complete, in the sense that for every existing interpolant there exists a solution for the constraints.

As for soundness, we need to show that $f(H_1, \dots, H_m) \subseteq H_{f(H_1,\dots,H_m)}$ for every solution, where $f$ is any of our operators. As a consequence, we have that $X \subseteq H_X$ is preserved through structural induction over the composition, from bottom to top. More specifically, we have that $X_1 \subseteq H_{X_1}, \dots, X_m \subseteq H_{X_m}$ implies $f(X_1, \dots, X_m) \subseteq H_{f(X_1,\dots,X_m)}$, by monotonicity of our operators. Hence, to obtain soundness, we need to guarantee $X \subseteq H_X$ for the bottom constraints as, then, every solution gives us valid halfspace interpolants.

As for completeness, we need to show that, after fixing $a_{f(X_1,\dots,X_m)}$ and $b_{f(X_1,\dots,X_m)}$, for every $H'_1, \dots, H'_m$ such that $f(H'_1, \dots, H'_m) \subseteq H_{f(X_1,\dots,X_m)}$, there exists a solution such that $H_{X_1} \supseteq H'_1, \dots, H_{X_m} \supseteq H'_m$. As a consequence, we have that for every set of valid interpolants, an assignment for the encoding can be constructed by structural induction, from top to bottom. For this purpose, we need two additional properties. First, we need that interpolants actually exist, which is guaranteed by strong interpolation. Second, we need that, for every bottom set $X$ and every candidate interpolant $H_X$ that includes $X$ strongly, the encoding for the bottom set admits an assignment.

For the encodings of intersection, Minkowski sum, and linear transformation, soundness holds, as we respectively show in Thm. 3.2.1, 3.2.2, and 3.2.3. For the encoding of an intersection, completeness holds for feasible intersections; for disjoint sets, it holds if the enclosing halfspace is either $\emptyset$ or $\mathbb{R}^n$.

**Theorem 3.2.1.** *For every solution to Eq. 3.18 it is necessary that*

$$H_{X_1} \cap \dots \cap H_{X_m} \subseteq H_{X_1 \cap \dots \cap X_m}. \tag{3.25}$$

*Moreover, for all halfspaces $H'_1 \cap \dots \cap H'_m \subseteq H_{X_1\cap\dots\cap X_m}$ such that either $a_{X_1\cap\dots\cap X_m} = 0$ (i.e., $H_{X_1\cap\dots\cap X_m}$ is either $\emptyset$ or $\mathbb{R}^n$) or $H'_1 \cap \dots \cap H'_m \neq \emptyset$, there exists a solution such that, for all $i = 1, \dots, m$, either $H_{X_i} = H'_i$ or $H_{X_i} = \mathbb{R}^n$.*

*Proof.* As for the first part, we have that $\rho_{H_{X_1} \cap \cdots \cap H_{X_m}} = \mathrm{cl}(\rho_{H_1} * \cdots * \rho_{H_m})$ [99, Cor. 16.4.4], as we show in Eq. 2.14. Then, by definition of closure we have the upper bound $\mathrm{cl}(\rho_{H_1} * \cdots * \rho_{H_m}) \leq \rho_{H_1} * \cdots * \rho_{H_m}$. Finally, by Eq. 3.18 that $a_{X_1} + \cdots + a_{X_m} = a_{X_1 \cap \cdots \cap X_m}$ and, since $\rho_{H_i}(a_{X_i}) = b_{X_i}$, we obtain the further upper bound $\rho_{H_1} * \cdots * \rho_{H_m}(a_{X_1 \cap \cdots \cap X_m}) \leq b_{X_1} + \cdots + b_{X_m} \leq b_{X_1 \cap \cdots \cap X_m}$. As a result, we have that $\rho_{H_{X_1} \cap \cdots \cap H_{X_m}}(a_{X_1 \cap \cdots \cap X_m}) \leq b_{X_1 \cap \cdots \cap X_m}$, which is equivalent to Eq. 3.25. As for the second part, let $H'_1 = \{x \colon \langle a'_1, x \rangle \leq b'_1\}, \ldots, H'_m = \{x \colon \langle a'_m, x \rangle \leq b'_m\}$. As for $\emptyset \neq H'_1 \cap \cdots \cap H'_m \subseteq H_{X_1 \cap \cdots \cap X_m}$, if $H_{X_1 \cap \cdots \cap X_m} \neq \mathbb{R}^n$, since $H'_1 \cap \cdots \cap H'_m$ is feasible and bounded in direction $a_{X_1 \cap \cdots \cap X_m}$ by $b_{X_1 \cap \cdots \cap X_m}$, we have that for some $\lambda_1, \ldots, \lambda_m \geq 0$, it holds that

$$\lambda_1 a'_1 + \cdots + \lambda_m a'_m = a_{X_1 \cap \cdots \cap X_m} \text{ and } \lambda_1 b'_1 + \cdots + \lambda_m b'_m \leq b_{X_1 \cap \cdots \cap X_m}, \tag{3.26}$$

by duality of linear programming. Let $a_{X_i} = \lambda_i a'_i$ and $b_{X_i} = \lambda_i b'_i$. First, by simple values substitution, Eq. 3.18 is satisfied. Second, for every $1 \leq i \leq m$, if $\lambda_i > 0$, we have that $H_{X_i} = \{x \colon \langle \lambda_i a'_i, x \rangle \leq \lambda_i b'_i\} = H'_i$, while, if $\lambda_i = 0$, we have that $H_{X_i} = \mathbb{R}^n$. Instead, if $H_{X_1 \cap \cdots \cap X_m} = \mathbb{R}^n$, then we have that $a_{X_1 \cap \cdots \cap X_m} = 0$ and $b_{X_1 \cap \cdots \cap X_m} \geq 0$; hence, for every $i = 1, \ldots, m$, take $a_{X_i} = 0$ and $b_{X_i} = 0$, for which $H_{X_i} = \mathbb{R}^n$. As for $H'_1 \cap \cdots \cap H'_m \subseteq H_{X_1 \cap \cdots \cap X_m} = \emptyset$, we have that $a_{X_1 \cap \cdots \cap X_m} = 0$ and $b_{X_1 \cap \cdots \cap X_m} < 0$, and (ii) that, by Farkas' lemma, there exist $\lambda_1, \ldots, \lambda_m \geq 0$ that satisfy Eq. 3.26. As above, we construct $H_{X_i} = \{x \colon \langle \lambda_i a'_i, x \rangle \leq \lambda_i b'_i\}$ and obtain that either $H_{X_i} = H'_i$ or $H_{X_i} = \mathbb{R}^n$. □

Notably, Eq. 3.18 might be incomplete for empty intersections and general enclosing halfspaces. For instance, consider $H'_1 = \{(x, y) \colon x \leq -1\}$ and $H'_2 = \{(x, y) \colon x \geq 1\}$,

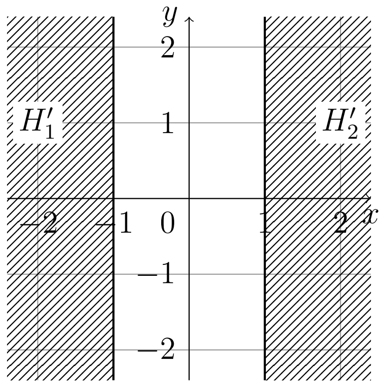


**Figure 3.4: Intersection of halfspaces for which Eq. 3.18 is incomplete.**

depicted in Fig. 3.4, and $a_{X_1 \cap X_2} = (1, 0)$ and $b_{X_1 \cap X_2} = 1$, giving the halfspace $H_{X_1 \cap X_2} = \{(x, y) \colon y \leq 1\}$. Finding a solution for Eq. 3.18 such that either $H_{X_1} = H'_1$ or $H_{X_1} = \mathbb{R}^n$

and either $H_{X_2} = H'_2$ or $H_{X_2} = \mathbb{R}^n$ amount to finding $\lambda_1, \lambda_2 \geq 0$ such that $a_{X_1} = \lambda_1(1,0)$, $b_{X_1} = -1\lambda_1$, $a_{X_2} = \lambda_2(-1,0)$, and $b_{X_2} = -1\lambda_2$, and that satisfy Eq. 3.18. Altogether, it amounts to finding a solution for

$$\begin{pmatrix}1\\0\end{pmatrix}\lambda_1 + \begin{pmatrix}-1\\0\end{pmatrix}\lambda_2 = \begin{pmatrix}0\\1\end{pmatrix}, \; -\lambda_1 - \lambda_2 \leq 1, \tag{3.27}$$

which does not exist; nevertheless, a solution exist if substituting $a_{X_1 \cap X_2} = (0,1)$ with $a_{X_1 \cap X_2} = 0$. In particular, thanks to Thm. 3.2.1, if $H_{X_1 \cap \dots \cap X_m} = \emptyset$, a solution exists if and only if disjoint interpolants exist; in other words, the encoding is complete for checking feasibility. As a consequence, for concatenations of several operators, completeness for checking feasibility holds if (i) $H_{f(X_1,\dots,X_m)}$ is either $\emptyset$ or $\mathbb{R}^n$ when $f(X_1,\dots,X_m) = \emptyset$ and (ii) the respective encoding is complete if either $f(X_1,\dots,X_m) \neq \emptyset$ or $H_{f(X_1,\dots,X_m)}$ is either $\emptyset$ or $\mathbb{R}^n$. Point (i) holds, in particular, for monotonic operators that always allow halfspace interpolation, where $H_{f(X_1,\dots,X_m)} = \emptyset$ whenever $f(X_1,\dots,X_m) = \emptyset$ (see Cor. 3.1.5.2). Point (ii) holds for intersection, by Thm. 3.2.1, for Minkowski sum and linear transformation, by the following theorems.

**Theorem 3.2.2.** *For every solution to Eq. 3.19 it is necessary that*

$$H_X + H_Y \subseteq H_{X+Y}. \tag{3.28}$$

*Moreover, for all halfspaces $H'$ and $H''$ such that $H' + H'' \subseteq H_{X+Y}$, if either $H' + H'' \neq \emptyset$ or $a_{X+Y} = 0$, then there exists a solution such that either $H_X = H'$ or $H_X = \mathbb{R}^n$ and either $H_Y = H''$ or $H_Y = \mathbb{R}^n$.*

*Proof.* As for the first part, $b_X + b_Y \leq b_{X+Y}$ implies that $\rho_{H_X}(a_X) + \rho_{H_Y}(a_Y) \leq \rho_{H_{X+Y}}(a_{X+Y})$, by definition of $H_X$, $H_Y$, and $H_{X+Y}$ (Eq. 3.23). Since $a_X = a_Y = a_{X+Y}$, we have that $\rho_{H_X}(a_{X+Y}) + \rho_{H_Y}(a_{X+Y}) \leq \rho_{H_{X+Y}}(a_{X+Y})$ and, therefore, that Eq. 3.28 holds. As for the second part, $a_X$ and $a_Y$ are clear by Eq. 3.19. If $\emptyset \neq H' + H'' \subseteq H_{X+Y} \neq \mathbb{R}^n$, we have $\rho_{H'}(a_{X+Y}) + \rho_{H''}(a_{X+Y}) \leq b_{X+Y}$ and, by taking $b_X = \rho_{H'}(a_{X+Y})$ and $b_Y = \rho_{H''}(a_{X+Y})$, we have $H_X = H'$ and $H_Y = H''$, since $a_{X+Y} \neq 0$. If $a_{X+Y} = 0$, it is necessary that $a_X = a_Y = 0$. If $H_X$ (resp. $H_X$) is non-empty, take $b_X = 0$ (resp. $b_Y = 0$), while if $H_X$ (resp $H_X$) is empty, choose any negative $b_X$ (resp. $b_Y$) that satisfy $b_X + b_Y \leq b_{X+Y}$. Note that, if both $H_X$ and $H_Y$ are non-empty, then $b_{X+Y} \geq 0$ and hence $b_X + b_Y \leq b_{X+Y}$ is satisfied. □

**Theorem 3.2.3.** *For every solution to Eq. 3.20 it is necessary that*

$$AH_X \subseteq H_{AX}. \tag{3.29}$$

*In addition, for all halfspaces $H'$ such that $AH' \subseteq H_{AX}$, if either $H' \neq \emptyset$ or $a_{AX} = 0$, there exists a solution such that either $H_X = H'$ or $H_X = \mathbb{R}^n$.*

*Proof.* Concerning the first part, $b_X \leq b_{AX}$ implies $\rho_{H_X}(a_X) \leq \rho_{H_{AX}}(a_{AX})$, by definition of $H_X$ and $H_{AX}$, which, by the constraint $a_X = A^\mathsf{T} a_{AX}$ in Eq. 3.20, is equivalent to $\rho_{H_X}(A^\mathsf{T} a_{AX}) \leq \rho_{H_{AX}}(a_{AX})$. By Eq. 2.11, we have $\rho_{AH_X}(a_{AX}) \leq \rho_{H_{AX}}(a_{AX})$, namely we have Eq. 3.29. Concerning the second part, take $a_X = A^\mathsf{T} a_{AX}$. If $H' \neq \emptyset$, we take $b_X = \rho_{H'}(a_X)$, which is bounded by $b_{AX}$, hence finite; we obtain that $H_X = H'$, if $a_X \neq 0$, $H_X = \mathbb{R}^n$, otherwise. If $a_{AX} = 0$ then we necessarily take $a_X = 0$. Then, if $H' = \emptyset$, we take any $b_X < 0$ and $b_X \leq b_{AX}$, obtaining $H_X = \emptyset$. Otherwise, if $H' \neq \emptyset$, then we have that $b_{AX} \geq 0$, hence we take $b_X = 0$, obtaining $H_X = \mathbb{R}^n$. □

We have established that, for abstractions consisting of Minkowski sum, linear transformations, and intersections of linear and compact sets, refining directions always exist and that can be computed using convex programming. In the following chapters, we instantiate our theory to the analysis of convex hybrid automata in Ch. 4, and extend it to the analysis of piece-wise affine systems in Ch. 5 and 6.

# 4 Counterexample-guided Refinement for Convex Hybrid Automata

We target the time-unbounded reachability analysis of convex hybrid automata (CHA), i.e., hybrid automata whose flow constraints consist of differential inclusions (on derivatives only) and all constraints (flow, guards, and invariants) are (possibly non-linear) closed convex sets, and the special cases of linear hybrid automata (LHA) and quadratic hybrid automata (QHA). A large class of systems belongs to CHA, e.g., timed systems with convex non-linear clock drifts, or can be approximated as CHA, e.g., systems with Gaussian disturbances truncated by elliptic sets. The reachability analysis of LHA has a long history [13], while for QHA or beyond only bounded reachability analysis has been explored [31; 42].

We show that (i) for every CHA halfspace interpolants suitable for refinement always exist and that (ii) they can be computed efficiently using convex optimization [28], in particular using linear programming for LHA and second-order conic programming for QHA. We implement a tool based of this technology and evaluate it on several linear and quadratic benchmarks, comparing (favorably) against PHAVer where that tool applies [52; 57], namely LHA. This gives the following new results. First, we enable the use of template polyhedra for the abstract interpretation and the abstraction refinement of CHA, thus enabling the efficient time-unbounded reachability analysis for the full class where efficient convex optimizers are available, namely QHA. Second, we achieve greater practical performance against the state-of-the-art techniques for the time-unbounded reachability of even LHA. We evaluate our tool on multiple scaling and linear and non-linear variants of three different benchmarks, namely Fischer's protocol [85], the TTEthernet protocol [25], and an adaptive cruise controller [73].

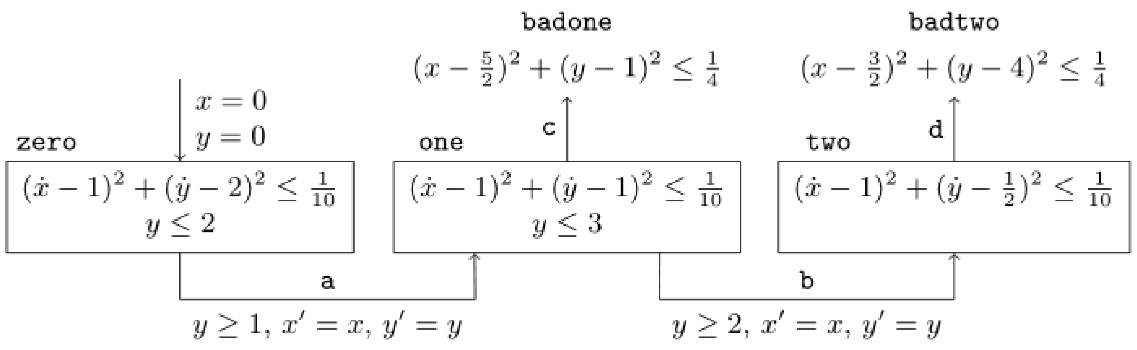
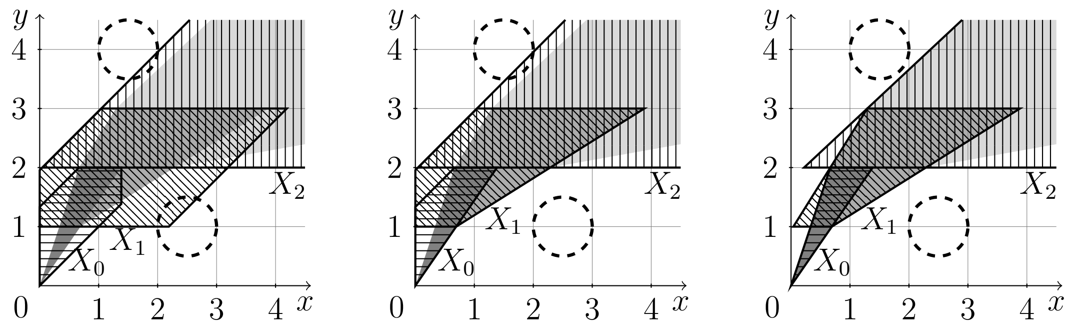

badone
$(x-\frac{5}{2})^2+(y-1)^2 \le \frac{1}{4}$
badtwo
$(x-\frac{3}{2})^2+(y-4)^2 \le \frac{1}{4}$
$x=0$
$y=0$
zero
c
one
d
two
$(\dot{x}-1)^2+(\dot{y}-2)^2 \le \frac{1}{10}$
$y \le 2$
$(\dot{x}-1)^2+(\dot{y}-1)^2 \le \frac{1}{10}$
$y \le 3$
$(\dot{x}-1)^2+(\dot{y}-\frac{1}{2})^2 \le \frac{1}{10}$
a
b
$y \ge 1,\ x'=x,\ y'=y$
$y \ge 2,\ x'=x,\ y'=y$

**Figure 4.1: A CHA with two variables $x$ and $y$, three good modes `zero`, `one`, and `two`, two bad modes `badone` and `badtwo`, and four switches `a`, `b`, `c`, and `d`. The good modes have three different relative speeds for $x$ and $y$ with an additional spherical drift. All invariants, the jump guards of `a` and of `b` are linear and the jump guards of `c` and of `d` are spherical.**

## 4.1 Motivating Example

Consider a system with two real-valued variables $x$ and $y$ whose dynamics follows some differential equation, which in turn is discontinuously switched by an automaton with three modes. Figure 4.1 shows such an example. The trajectory starts in the origin and enters mode `zero` and follows any differential equation whose derivative is $\dot{x} = 1$ and $\dot{y} = 2$ with possibly some drift in the ball of radius $10^{-\frac{1}{2}}$ around this value. The invariant allows the trajectory to stay in mode `zero` as long as $y \le 2$. The trajectory can take `a` if $y \ge 1$ and switch to mode `one`, where the derivative of $y$ halves. The dynamics continues similarly on mode `one`, switch `c`, and mode `two`, and similarly can take a switch to `badone` and `badtwo` when the respective guards are satisfied. We know that there does not exists a trajectory that leads to one of the bad modes, namely the system is *safe*. We want to prove it automatically by means of template polyhedra.

The set of states that are respectively reachable on modes `zero`, `one`, and `two` are the cones spanned by the points that enter the mode and take any possible trajectory, as respectively depicted in Fig. 4.2 in three shades on gray. We abstract the whole systems by representing each of these sets using template polyhedra. But first, we need to discover



(a) octagonal abstraction (b) w/o path to `badone` (c) w/o path to `badtwo`

**Figure 4.2: Template-polyhedral abstraction refinement for the CHA in Fig. 4.1. In dark gray, gray, and light gray the points reachable on the modes `zero`, `one`, and `two`, resp., and the striped polyhedra $X_0$, $X_1$, and $X_2$ are the resp. template polyhedra. The lower and the upper dashed circles are, resp., the guards of the switches `c` and `d` to the bad modes. The variant (a) show the octagonal abstraction, and (b) and (c) show resp. the results of the templates obtained after refinement of the paths to `badone` and then to `badtwo`.**

a suitable template. In fact, different templates produce different abstractions and not all of them can prove safety. Figure 4.2 shows three different such abstractions (striped polyhedra), but (a) and (b) hit the guards (dashed circled) to the bad modes while only (c) accomplishes the task. Our goal is to construct a good template like in (c).

We begin with abstraction (a) which uses the octagonal template, i.e., the 8 orthogonal directions to the facets of an octagon. The abstract interpreter will produce several abstract paths (sequences of pairs of modes and polyhedra interleaved by switches) among which will occur the path `zero`, `a`, `one`, `c`, `badone`, for the regions $X_0, X_1 \subseteq \mathbb{R}^n$ where $X_0 = \mathsf{init}_{\mathtt{zero}}$ abstracts the flow on `zero`, and $X_1 = \mathsf{post}_{\mathtt{a}}(X_0)$ abstracts the flow on `one` (see Fig. 4.2a). This path reaches a bad mode, but it is spurious, namely it does not have a concrete counterpart. We prove it by computing a sequence of halfspace interpolants, i.e., two halfspaces $H_0$ and $H_1$ such that $\mathsf{init}_{\mathtt{zero}} \subseteq H_0$ and $\mathsf{post}_{\mathtt{a}}(H_0) \subseteq H_1$ and $H_1$ does not intersect with the guard of the switch `c` (see Fig. 4.3b). The outward pointing directions $d_0$ and $d_1$ of $H_0$ and $H_1$ are the directions that generalize and eliminate all counterexamples with the switching sequence `zero`, `a`, `one`, `c`, `badone` (see Fig. 4.3c). We add them to the template and we recompute the abstraction, obtaining a necessarily different counterexample (see Fig. 4.2b). We repeat and eventually obtain Fig. 4.2c, finally proving the safety of the hybrid automaton.

In the next section we define the modeling and the (template-polyhedral) abstraction framework for CHA. In Sec. 4.3 we present our interpolant-based refinement technique and in Sec. 4.4 we phrase it as a convex optimization problem. In Sec. 4.5 we instantiate it to QHA and in Sec. 6.6 we show our experimental results.

## 4.2 Abstractions of Convex Hybrid Automata using Template Polyhedra

Hybrid automata extend finite automata adding constraints on the (discrete and continuous) dynamics of a set of real variables (see Sec. 2.1). Convex hybrid automata (CHA) are the class whose constraint define non-linear convex sets that exclusively constrain either variables or variable derivatives, as it is the case for the well-know class of linear hybrid automata (LHA) [63], which is thus generalized by CHA.

**Definition 2** (Convex hybrid automata)**.** A *convex hybrid automaton* $\mathcal{H}$ is a hybrid automaton where all constraints define closed convex sets and all flow constraints are constant and do not contain zero, i.e., for every $v \in V$ and all $x, y \in \mathbb{R}^n$, $0 \notin F_v(x) = F_v(y)$.

In this chapter, we treat every $F_v$ as closed convex sets in $\mathbb{R}^n$ and $J_e$ as closed convex sets in $\mathbb{R}^{2n}$ (in other words, as relations in $\mathbb{R}^n \times \mathbb{R}^n$). When a control path is clear from the context, we abbreviate any object indexed by $v_i$ or $e_i$ as the same object indexed by $i$, e.g., we abbreviate $F_{v_i}$ as $F_i$. The semantics associates modes to points $x \in \mathbb{R}^n$. For every two points $x, x' \in \mathbb{R}^n$, for every control mode $v \in V$ we say that $x'$ is a *$v$-successor* of $x$ if there exists a derivable solution $\psi\colon \mathbb{R} \to \mathbb{R}^n$ and a time delay $\delta \geq 0$ such that $\psi(0) = x$, $\psi(\delta) = x'$, and for all $0 \leq \gamma \leq \delta$ it holds that $\dot{\psi}(\gamma) \in F_v$ and $\psi(\gamma) \in I_v$, and for every control switch $e \in E$ we say that $x'$ is an *$e$-successor* of $x$ if $(x, x') \in J_e$.

**Definition 3** ($\mathcal{H}$-feasibility)**.** A finite control path $v_0, e_1, v_1, \ldots, e_k, v_k$ is $\mathcal{H}$-feasible if for some $x_0, x_0', x_1, x_1', \ldots, x_k, x_k' \in \mathbb{R}^n$ it holds that $x_0 \in Z_0$, and for all $0 \leq i \leq k$, $x_i'$ is a $v_i$-successor of $x_i$ and $x_i$ is a $e_i$-successor of $x_{i-1}'$.

The semantics of $\mathcal{H}$ is the maximal set of $\mathcal{H}$-feasible paths. A mode $v \in V$ is *reachable* if there exists an $\mathcal{H}$-feasible control path whose last mode is $v$, and a point $x' \in \mathbb{R}^n$ is reachable on $v$ if $x'$ is the last point of a sequence as in Def. 3.

The abstraction associates modes to regions of $\mathbb{R}^n$ into abstract paths whose elements are related by the init and post operator of an abstraction structure $\mathcal{A}$.

**Definition 4** (Abstraction structure)**.** An abstraction structure $\mathcal{A}$ for the CHA $\mathcal{H}$ consists of an init operator $\mathsf{init}_v \in \wp(\mathbb{R}^n)$ for every $v \in V$ and of a post operator $\mathsf{post}_e \colon \wp(\mathbb{R}^n) \to \wp(\mathbb{R}^n)$ for every $e \in E$.

Similarly as for $\mathcal{H}$, a control path with an abstract counterpart is called $\mathcal{A}$-feasible.

**Definition 5** ($\mathcal{A}$-feasibility)**.** A finite control path $v_0, e_1, v_1, \dots, e_k, v_k$ is $\mathcal{A}$-feasible if for some non-empty sets $X_0, X_1, \dots, X_k \subseteq \mathbb{R}^n$ holds that $X_0 = \mathsf{init}_0$ and for all $1 \leq i \leq k$, $X_i = \mathsf{post}_i(X_{i-1})$.

An $\mathcal{A}$-feasible path is *genuine* if it is also $\mathcal{H}$-feasible, and *spurious* otherwise. An abstraction structure $\mathcal{A}$ is *sound* if all $\mathcal{H}$-feasible control paths are $\mathcal{A}$-feasible.

The *support function* [99] in direction $d \in \mathbb{R}^n$ of a convex set $X \subseteq \mathbb{R}^n$ is

$$\rho_X(d) = \sup\{d \cdot x \mid x \in X\}. \tag{4.1}$$

The support function of $X$ characterizes the template polyhedron [103; 59] of $X$ for the template $\Delta \subseteq \mathbb{R}^n$ (a finite set). We call it the $\Delta$-*polyhedron* of $X$, that is

$$\bigcap_{d \in \Delta} \{x \in \mathbb{R}^n \mid d \cdot x \leq \rho_X(d)\}. \tag{4.2}$$

We aim at computing template polyhedra for the (continuous) flow and the (discrete) jump post operators (and their compositions) of the hybrid automaton. The *flow operator* of mode $v \in V$ gives the points reachable by time elapse on $v$:[1]

$$\mathrm{flow}_v(X) = (X + \operatorname{coni} F_v) \cap I_v. \tag{4.3}$$

The *jump operator* of switch $e \in E$ gives the points reachable through $e$:

$$\mathrm{jump}_e(X) = \begin{bmatrix} 0_{n\times n} & I_n \end{bmatrix} \left( \left( \begin{bmatrix} I_n \\ 0_{n\times n} \end{bmatrix} X + \begin{bmatrix} 0_{n\times n} \\ I_n \end{bmatrix} \mathbb{R}^n \right) \cap J_e \right). \tag{4.4}$$

Flow and jump operators are an exact symbolical characterization for the semantics of CHA, and follow as an extension of the symbolic analysis of LHA [63; 6].

[1]For $X \subseteq \mathbb{R}^n$, $\operatorname{coni} X$ denotes the conical hull $\{0\} \cup \{\alpha x \mid \alpha > 0 \wedge x \in X\}$.

**Lemma 4.2.1.** *For every CHA $\mathcal{H}$ and every set $X \subseteq \mathbb{R}^n$ it holds that (i) $x' \in \mathrm{flow}_v(X)$ if and only if $x'$ is a $v$-successor of some $x \in X$ for every control mode $v \in V$ and (ii) $x' \in \mathrm{jump}_e(X)$ if and only if $x'$ is a $e$-successor of some $x \in X$ for every control switch $e \in E$.*

The exact symbolic analysis of CHA has in general high complexity, as it requires eliminating quantifiers, and possibly from formulae that contain non-linear constraints. For this reason we approximate them using template polyhedra.

The template-polyhedral abstraction computes the template polyhedra of the flow and jump operators above and, in our definition, using a different template for each mode, given by the *precision function* $\mathsf{prec}\colon V \to \wp(\mathbb{R}^n)$.

**Definition 6** (Template-polyhedral abstraction)**.** The template-polyhedral abstraction for the CHA $\mathcal{H}$ and the precision function $\mathsf{prec}\colon V \to \wp(\mathbb{R}^n)$ is the abstraction structure where the init operator $\mathsf{init}_v$ is the $\mathsf{prec}(v)$-polyhedron of $\mathrm{flow}_v(Z_v)$, and the post operator $\mathsf{post}_e(X)$ is the $\mathsf{prec}(t)$-polyhedron of $\mathrm{flow}_t \circ \mathrm{jump}_e(X)$ where $t \in V$ is the destination of $e$.

It is well-know that the template-polyhedral abstraction constructs a conservative over-approximation for linear systems [103], and the same holds for CHA.

**Theorem 4.2.2.** *For every CHA $\mathcal{H}$ and every precision function $\mathsf{prec}$ the template-polyhedral abstraction for $\mathcal{H}$ and $\mathsf{prec}$ is sound.*

*Proof.* By induction over any $\mathcal{A}$-feasible path, for the sets $X_0, X_1, \dots,$ of a template-polyhedral abstraction for $\mathcal{H}$. As for the base case, $X_0 = \mathsf{init}_v \supseteq \mathrm{flow}_0(Z_0)$, which includes the $v_0$-successors of $Z_0$ (Lem. 4.2.1), hence the points reachable by $v_0$. As for the inductive case, $X_{i+1} = \mathsf{post}_{i+1} \supseteq \mathrm{flow}_{i+1} \circ \mathrm{jump}_{i+1}(X_i)$, which includes the $v_{i+1}$-successors of the $e_{i+1}$-successors of $X_i$ (Lem. 4.2.1). Hence, if $X_i$ includes the points reachable by $v_0 e_1 \dots v_i$ then $X_{i+1}$ must include those reachable by $v_0 e_1 \dots v_{i+1}$. □

The obvious difficulty is in finding a precision function that is suitable for proving or disproving reachability. In the next section, we show how to form one such automatically by means of counter-example guided abstraction refinement.

## 4.3 Counterexample-guided Refinement using Halfspace Interpolants

A counter-example guided abstraction refinement (CEGAR) loop [44] for a hybrid automaton $\mathcal{H}$ and a set of bad modes $T$ consists of an abstractor and a refiner interacting with each other. At each iteration $i$, the *abstractor* takes an abstraction structure $\mathcal{A}_i$ and attempts to construct the finite state machine that recognizes all $\mathcal{A}_i$-feasible paths. If it terminates and it does not find a counterexample, i.e., a path leading to a bad mode, then it returns `no`. Otherwise, it passes $\mathcal{A}_i$ and a set of counterexamples $W_i$ to the refiner. The *refiner* attempts to construct an abstraction structure $\mathcal{A}_{i+1}$ that refines $\mathcal{A}_i$ and eliminates all counterexamples in $W_i$. If it fails, then it reports `yes` and a set $\bar{W}_i \subseteq W_i$ of genuine counterexamples. Otherwise, it passes $\mathcal{A}_{i+1}$ to the abstractor.

The above procedure is sound (upon termination), provided $\mathcal{A}_i$ is sound, in the sense that if it reports `no` then no mode in $T$ is reachable. It is complete (upon termination), namely if it reports `yes` then some mode in $T$ is reachable, if it returns an abstraction $\mathcal{A}_{i+1}$ that is locally complete w.r.t. $W_i$ when one exists.

**Local completeness** An abstraction structure $\mathcal{A}$ for the CHA $\mathcal{H}$ is locally complete w.r.t. the set $W$ of control paths of $\mathcal{H}$ if all $\mathcal{H}$-infeasible control paths in $W$ are $\mathcal{A}$-infeasible.

Moreover, if it ensures local completeness w.r.t. $\cup\{W_j | 0 \leq j \leq i\}$, then it ensures progress of the procedure if the counterexamples are given one by one.

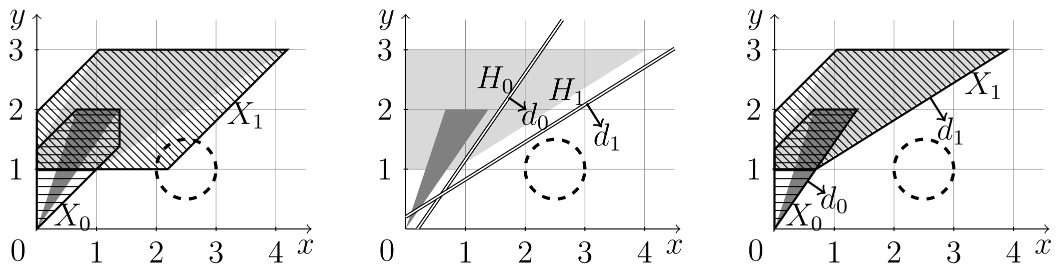
Whenever we find a spurious counterexample, we augment the precision of the modes along the path with additional template directions, so to make it $\mathcal{A}$-infeasible. First of all, we start with finding a sequence of Craig's interpolants and only Craig's interpolants that are halfspaces [2]. Formally, let $w = v_0, e_1, v_1, \ldots, e_k, v_k$ be a control path of $\mathcal{H}$, then a sequence of *halfspace interpolants* for $w$ is a sequence of sets $H_0, H_1, \ldots, H_k \subseteq \mathbb{R}^n$ such that each element is either the universe, a closed halfspace, or the empty set and

$$\mathrm{flow}_0(Z_0) \subseteq H_0, \mathrm{flow}_1 \circ \mathrm{jump}_1(H_0) \subseteq H_1, \ldots, \mathrm{flow}_k \circ \mathrm{jump}_k(H_{k-1}) \subseteq H_k, \tag{4.5}$$

and $H_k \subseteq \emptyset$. If such sequence exists, then the path is clearly $\mathcal{H}$-infeasible. Conversely, it is not trivial that for every $\mathcal{H}$-infeasible path such sequence exists.

**Lemma 4.3.1.** *For every CHA $\mathcal{H}$, where either (i) all constraints are compact or (ii) all constraints are polyhedral, and every control path $w$ of $\mathcal{H}$, it holds that every control path $w$ is $\mathcal{H}$-infeasible if and only if there exists a sequence $H_0, H_1, \dots, H_k \subseteq \mathbb{R}^n$ of halfspace interpolants for $w$ as in Eq. 4.5.*

*Proof.* The if direction follows from the soundness of $\mathrm{flow}_v$ and $\mathrm{jump}_e$ (Lem. 4.2.1). As for the only if direction, we let $v_0 e_1 \dots v_k$ be a control path and show that, for every $i = 1, \dots, k$, $f_i = \mathrm{flow}_i \circ \mathrm{jump}_i$ admits strong halfspace interpolation, with the respective operand $X_i = \mathrm{flow}_{i-1} \circ \mathrm{jump}_{i-1}(X_{i-1})$, for $i \geq 0$, and $X_1 = \mathrm{flow}_0(Z_0)$. First, we break down $\mathrm{flow}_v \circ \mathrm{jump}_e$ into six operations $X^{(1)} = [I_n \; 0_{n \times n}]^\mathsf{T} X^{(0)}$, $X^{(2)} = X^{(1)} + [0_{n \times n} \; I_n]^\mathsf{T} \mathbb{R}^n$, $X^{(3)} = X^{(2)} \cap J_e$, $X^{(4)} = [0_{n \times n} \; I_n] X^{(3)}$, $X^{(5)} = X^{(4)} + \operatorname{coni} F_v$, and $X^{(6)} = X^{(5)} \cap I_v$. Second, we show that, if (i) all constraints and $X^{(0)}$ are compact, then $X^{(1)}, \dots, X^{(6)}$ are closed and $X^{(6)}$ is compact. Since $X^{(0)}$ is compact, $X^{(1)}$ is compact [99, Thm. 9.1]. Since $X^{(1)}$ is compact and $[0_{n \times n} \; I_n]^\mathsf{T} \mathbb{R}^n$ is closed, then $X^{(2)}$ is closed [99, Cor. 9.1.2]. Since $J_e$ is compact, then then $X^{(3)}$ and $X^{(4)}$ are compact, too. Since $F_v$ is compact and does not contain the origin, $\operatorname{coni} F_v$ is closed [99, Cor. 9.6.1], and, since $X^{(4)}$ is compact, $X^{(5)}$ is closed [99, Cor. 9.1.2]. Finally, since $I_v$ is compact, then $X^{(6)}$ is compact. Instead, if (ii) all constraints and $X^{(0)}$ are polyhedral, then $X^{(1)}, \dots, X^{(6)}$ are polyhedral, as every linear transformation [99, Thm. 19.3], Minkowski sum [99, Cor. 19.3.2], and intersection of polyhedra are necessarily polyhedral. Third, we show that the premise that the argument $X_i$ is, resp., compact or polyhedral, holds for every of $f_1, \dots, f_k$. Let $X^{(4)} = Z_0$, then, from the argument above, $X_1 = X^{(6)}$ is, resp., compact or polyhedral. Let $X^{(0)} = X_i$, for $i \geq 1$, and assume it, resp., compact or polyhedra.; by the statements above, $X_{i+1} = X^{(6)}$ is, resp., compact ot polyhedral. By induction, if (i) all constraints are compact, then $X_1, \dots, X_k$ are compact, if (i) all constraints are polyhedral, then $X_1, \dots, X_k$ are polyhedral. Fourth, we show that each of the six operations composing admit strong halfspace interpolation. As for the Minkowski sums and the linear transformations, it follows from, resp., Thm. 3.1.3 and 3.1.4. As for the intersections, if (i) all constraints are compact, $X^{(1)}$ and $X^{(4)}$ are closed, $I_v$ and $J_e$ are compact, hence, by Cor. 3.1.1.2, $X^{(1)} \cap I_v$ and $X^{(4)} \cap J_e$ admit halfspace interpolation; if (ii) all constraints are polyhedral, $X^{(1)}$ and $X^{(4)}$ are polyhedral and, hence, the intersections admit halfspace interpolation by Thm. 3.1.2. As a consequence, by Cor. 3.1.5.1, every $f_i$, for $i = 1, \dots, k$, admits strong halfspace interpolation. Finally, the claim of the lemma follows from Cor. 3.1.5.2. □



(a) octagonal abstraction (b) halfspace interpolants (c) refined abstraction

**Figure 4.3: Refinement for the control path `zero`, `a`, `one`, `b`, `badone` of the CHA in Fig. 4.1. In dark gray, the points reachable on mode `zero`. In (a), (b), and (c), in light gray are the points reachable on mode `one` resp. from $X_0$, $H_0$, and $X_0$. In (a) the spurious path, in (b) the interpolants, and in (c) the abstraction with the outward pointing directions.**

Indeed, existence relies on the conditions we discussed in Sec. 3.1. Computing interpolants is the subject of the next section.

The refining directions are the outward pointing directions of the halfspace interpolants, respectively for each mode along the path. In fact, it is enough to observe that every abstraction we obtain after adding such directions also satisfy

$$\mathsf{init}_0 \subseteq H_0, \mathsf{post}_1(H_0) \subseteq H_1, \dots, \mathsf{post}_k(H_{k-1}) \subseteq H_k. \tag{4.6}$$

Figure 4.3 shows such an example. The path is the one leading to `badone` from the CHA of Fig. 4.1, which is spurious with octagonal template (see Fig. 4.3a), and, in fact, a sequence $H_0$ and $H_1$ of halfspace interpolants exists (see Fig. 4.3b). The halfspace $H_1$ is disjoint from the guard of $\mathsf{c}$ (dashed circle) and includes the points reachable from $H_0$ (light gray), which in its turn includes the points reachable from $Z_{\mathtt{zero}}$, i.e., $\mathrm{flow}_{\mathtt{zero}}(Z_{\mathtt{zero}}) \subseteq H_0$, $\mathrm{flow}_{\mathtt{one}} \circ \mathrm{jump}_{\mathsf{a}}(H_0) \subseteq H_1$, and $\mathrm{jump}_{\mathsf{c}}(H_1) \subseteq \emptyset$. Taking the supporting halfspaces in the same directions preserves these inclusions, hence adding $d_0$ to $\mathsf{prec}(\mathtt{zero})$ and $d_1$ to $\mathsf{prec}(\mathtt{one})$ causes $\mathsf{init}_{\mathtt{zero}} \subseteq H_0$, $\mathsf{post}_{\mathsf{a}}(H_0) \subseteq H_1$, and $\mathsf{post}_{\mathsf{c}}(H_1) \subseteq \emptyset$. Thus $d_0$ and $d_1$ eliminate the counterexample, and regardless of whether $\mathsf{prec}$ contains further directions (see Fig. 4.3c).

**Definition 7** (Template-polyhedral refinement)**.** Let $\mathcal{H}$ be a CHA and let $w = v_0, e_1, v_1, \dots, e_k, v_k$ be a control path. Define the precision function $\mathsf{prec}$ such that for some (if one exists) sequence of halfspace interpolants $H_0, H_1, \dots, H_k \subseteq \mathbb{R}^n$ for $w$ as in Eq. 4.5 then for all $0 \leq i \leq k$ set $d_i \in \mathsf{prec}(v_i)$ where $d_i$ is the outward pointing direction of $H_i$. We define the template-polyhedral refinement for $\mathcal{H}$ and $w$ as the template-polyhedral abstraction for $\mathcal{H}$ and $\mathsf{prec}$.

Local completeness w.r.t. a single path easily generalizes to local completeness w.r.t. multiple paths by taking the union of the discovered directions.

**Theorem 4.3.2.** *For every CHA $\mathcal{H}$, where either (i) all constraints are compact or (ii) all constraints are polyhedral, and every set $W$ of finite control paths of $\mathcal{H}$ the union*[2] *over all $w \in W$ of the template-polyhedral refinements for $\mathcal{H}$ and $w$ is locally complete w.r.t. $W$.*

*Proof.* By Lem. 4.3.1, for every $\mathcal{H}$-infeasible control path $v_0e_1 \dots v_k \in W$ there exists a sequence of halfspace interpolants $H_0, \dots, H_k$. Let $d_0, \dots, d_k$ be the respective outward pointing directions, and let $X_i = \mathsf{post}_i \circ \dots \circ \mathsf{post}_1(\mathsf{init}_0)$ for, respectively, the templates $\{d_0\}, \dots, \{d_i\}$. First, we have that $\mathsf{init}_0 \subseteq H_0$, and then that $X_i \subseteq H_i$, for every $i = 1, \dots, k$. Since $H_K = \emptyset$, then the control path is $\mathcal{A}$-infeasible. □

Summarizing, we search for abstract counterexamples and we accumulate all outward pointing directions of the respective halfspace interpolants. If either the abstractor finds a fixpoint or interpolation fails, then we obtain a sound and complete answer. In the following section, we show how to compute init and post operators and sequences of halfspace interpolants by using convex optimization.

[2]The union of the abstractions $\mathcal{A}_1, \dots, \mathcal{A}_i$ for $\mathcal{H}$ and resp. the precisions $\mathsf{prec}_1, \dots, \mathsf{prec}_i$ is the abstraction for $\mathcal{H}$ and the precision $\lambda v.\mathsf{prec}_1(v) \cup \dots \cup \mathsf{prec}_i(v)$.

## 4.4 Craig's Interpolation as Convex Optimization

The support function is a central actor both in abstraction, as it defines template polyhedra, and refinement, as it gives a powerful formalism to talk about inclusion in halfspaces and separation of convex sets. In either case, the sets we deal with are arbitrary compositions of flow and jump operators, which in their turn are compositions of Minkowski sums, linear transformations, conical combinations, and intersections. We characterize the support functions of such operations as convex programs, with the aim of characterizing abstraction and refinement as convex programs.

We present a characterization of support functions that is compositional for the set operations above. The *classic* support function representation framework[3] offers a very similar machinery [59], but it suffers from the following shortcomings. First, it requires the operand sets in Minkowski sums and intersections to be compact (i.e., closed and bounded) and boundedness cannot be easily relaxed, e.g., $\rho_{\mathbb{R}^n}(d) + \rho_{\emptyset}(d) = +\infty - \infty$ while $\rho_{\mathbb{R}^n+\emptyset}(d) = -\infty$ for every $d \neq 0$. Since we aim at time-unbounded reachability, it would be too restrictive to assume boundedness. Second, substituting boundedness with nonemptiness might cause uncorrect results, e.g., for the sets $A = \{(x, y) \mid x \leq -1\}$, $B = \{(x, y) \mid x \geq 1\}$, and the direction $c = (0, 1)$ we obtain $\inf\{\rho_A(c - a) + \rho_B(a)\} = +\infty$, while $\rho_{A\cap B}(c) = -\infty$. We relax both the assumptions of boundedness and nonemptiness by characterizing the support function $\rho_X(d)$ with a convex program

$$\begin{aligned} &\text{minimize} && \bar{\rho}_X(\lambda) \\ &\text{subject to} && (\lambda, d) \in \Lambda_X, \end{aligned} \tag{4.7}$$

with objective function $\bar{\rho}_X \colon \mathbb{R}^m \to \mathbb{R}$ and constraint $\Lambda_X \subseteq \mathbb{R}^{m+n}$. The minimum of $\bar{\rho}_X(\lambda)$ over $\lambda$ characterizes $\rho_X(d)$ for directions in which $X$ is bounded, while $\Lambda_X$ characterizes boundedness. This is encapsulated by the notion of duality.

**Duality** Let $X \subseteq \mathbb{R}^n$ be a nonempty closed convex set. The convex program of Eq. 4.7 is *dual* to $\rho_X$ if for all $d \in \mathbb{R}^n$ it holds that

(i) $\rho_X(d) = +\infty$ if and only if there does not exist $\lambda$ such that $(\lambda, d) \in \Lambda_X$,

(ii) $\rho_X(d) < +\infty$ if and only if $\rho_X(d) = \inf\{\bar{\rho}_X(\lambda) \mid (\lambda, d) \in \Lambda_X\}$.

[3] $\rho_{X+Y}(d) = \rho_X(d) + \rho_Y(d)$, $\rho_{MX}(d) = \rho_X(M^\mathsf{T} d)$, and $\rho_{X\cap Y}(d) = \inf\{\rho_X(a) + \rho_Y(d - a)\}$.

We define inductive rules for constructing dual convex programs for the support functions of set operations, provided dual convex programs for their operands (whose instantiation for sets defined by symbolic constraints is subject of Sec. 4.5):

$$\begin{aligned} \bar\rho_{X+Y}(\lambda,\mu) &= \bar\rho_X(\lambda)+\bar\rho_Y(\mu), \\ \Lambda_{X+Y} &= \{(\lambda,\mu,d) \mid (\lambda,d)\in\Lambda_X, (\mu,d)\in\Lambda_Y\}, \end{aligned} \tag{4.8}$$

$$\begin{aligned} \bar\rho_{AX}(\lambda) &= \bar\rho_X(\lambda), \\ \Lambda_{AX} &= \{(\lambda,d) \mid (\lambda,A^{\mathsf{T}}d)\in\Lambda_X\}, \end{aligned} \tag{4.9}$$

$$\begin{aligned} \bar\rho_{\operatorname{coni}X}(\lambda) &= 0, \\ \Lambda_{\operatorname{coni}X} &= \{(\lambda,d) \mid \bar\rho_X(\lambda)\le 0, (\lambda,d)\in\Lambda_X\}, \end{aligned} \tag{4.10}$$

$$\begin{aligned} \bar\rho_{X\cap Y}(\lambda,\mu) &= \bar\rho_X(\lambda)+\bar\rho_Y(\mu), \text{ and} \\ \Lambda_{X\cap Y} &= \{(\lambda,\mu,a,d) \mid (\lambda,a)\in\Lambda_X, (\mu,d-a)\in\Lambda_Y\}. \end{aligned} \tag{4.11}$$

Nevertheless, duality is not sufficient to characterize operations producing the empty set. Considering the examples above, the constraint $\Lambda_{\mathbb{R}^n+\emptyset}$ is infeasible for every direction $d\neq 0$ and the constraint $\Lambda_{A\cap B}$ is infeasible for direction $c$, contradicting (i). However, it suffices that the convex program is unbounded for at least $d=0$, providing an alternative for deciding emptiness beforehand.

**Alternativity** The convex program of Eq. 4.7 is *alternative* to $\rho_\emptyset$ if for every $\epsilon<0$ there exists $(\lambda,0)\in\Lambda_\emptyset$ such that $\bar\rho_\emptyset(\lambda)\le\epsilon$.

Altogether, we compute the support of $X$ in direction $d$ as follows. We decide whether there exists a negative solution in direction 0. If so we return $-\infty$, otherwise we decide whether $\Lambda_X$ is infeasible in direction $d$. If so we return $+\infty$, otherwise we solve the convex program. This is permitted on any combination of the set operations above, as our construction preserves duality and alternativity. To show this, we first extend the encoding of Sec. 3.2 with constraints for the conical hull operator, with the following lemma.

**Lemma 4.4.1.** *Let $X$ be a non-empty convex set in $\mathbb{R}^n$, $a_{\operatorname{coni}X}, a_X\in\mathbb{R}^n$ and $b_{\operatorname{coni}X}, b_X\in\mathbb{R}^n$, and let $H_X=\{x\colon \langle a_X,x\rangle\le b_X\}$ and $H_{\operatorname{coni}X}=\{x\colon \langle a_{\operatorname{coni}X},x\rangle\le b_{\operatorname{coni}X}\}$. If $X\subseteq H_X$ and*

$$a_X = a_{\operatorname{coni}X} \qquad b_X\le 0 \qquad 0\le b_{\operatorname{coni}X} \tag{4.12}$$

*is satisfied, then $\operatorname{coni}X\subseteq H_{\operatorname{coni}X}$. Moreover, for every $a_{\operatorname{coni}X}$ and $b_{\operatorname{coni}X}$ such that $\operatorname{coni}X\subseteq H_{\operatorname{coni}X}$, there exists $a_X$ and $b_X$ such that $X\subseteq H_X$ and Eq. 4.12 is satisfied.*

*Proof.* For every direction $d \in \mathbb{R}^n$ and, resp., the homogeneous halfspace $H = \{x \colon \langle d, x\rangle \leq 0\}$, we have that, if $X \subseteq H$, then $\operatorname{coni} X \subseteq H$ [99, Cor. 11.7.2]. Hence, for $d = a_X = a_{\operatorname{coni} X}$, $H$ as above, and Eq. 4.12 satisfied, we have that $H_X \subseteq H \subseteq H_{\operatorname{coni} X}$. Consequently, $X \subseteq H_X$ implies that $X \subseteq H$ and, hence, that $\operatorname{coni} X \subseteq H \subseteq H_{\operatorname{coni} X}$. As for the second part, take $a_X = a_{\operatorname{coni} X}$ and $b_X = 0$. First, since $\operatorname{coni} X \subseteq H_{\operatorname{coni} X}$, we have that $b_{\operatorname{coni} X} \geq 0$, and Eq. 4.12 is satisfied. Second, we also have that $\operatorname{coni} X \subseteq H_X$ and, since $X \subseteq \operatorname{coni} X$ [99, Cor. 2.6.3], we have that $X \subseteq H_X$. $\square$

As a result, we have that $\bar{\rho}_X$ and $\Lambda_X$ follow the construction of the encoding of Sec. 3.2 together with Eq. 4.12. In particular, for every set $X$ we have that, if $d = a_X$, then $\bar{\rho}_X(\lambda) \leq b_X$ and $\Lambda_X(\lambda, a_X)$, where $a_X$ and $b_X$ correspond to those of Sec. 3.2. More concretely, for the constraints $\Lambda_X$ we have

$$\Lambda_{X+Y}(\lambda, \mu, a_{X+Y}) \iff \Lambda_X(\lambda, a_X), \Lambda_Y(\mu, a_Y), a_X = a_Y = a_{X+Y}, \tag{4.13}$$

$$\Lambda_{AX}(\lambda, a_{AX}) \iff \Lambda_X(\lambda, a_X), a_X = A^{\mathsf{T}} a_{MX}, \tag{4.14}$$

$$\Lambda_{\operatorname{coni} X}(\lambda, a_{\operatorname{coni} X}) \iff \Lambda_X(\lambda, a_X), a_X = a_{\operatorname{coni} X}, b_X \leq 0 \tag{4.15}$$

$$\Lambda_{X\cap Y}(\lambda, \mu, a_X, a_{X\cap Y}) \iff \Lambda_X(\lambda, a_X), \Lambda_Y(\mu, a_Y), a_X + a_Y = a_{X\cap Y}, \tag{4.16}$$

while for the objective function $b_X$ we have $b_{X+Y} \geq \bar{\rho}_{X+Y}(\lambda, \mu)$, $b_{AX} \geq \bar{\rho}_{AX}(\lambda)$, $b_{\operatorname{coni} X} \geq \bar{\rho}_{\operatorname{coni} X}(\lambda)$, and $b_{X\cap Y} \geq \bar{\rho}_{X\cap Y}(\lambda)$. Consequently, duality and alternativity follow from the soundness and completeness of the encoding, as shown in Sec. 3.2. In particular, completeness relies on the existance of strong halfspace interpolants which, for intersections, holds for polyhedral and compact intersections, as shown in Sec. 3.1.

**Lemma 4.4.2.** *Let $X, Y \subseteq \mathbb{R}^n$ be closed convex sets. If the convex programs for $\bar{\rho}_X, \Lambda_X$ and $\bar{\rho}_Y, \Lambda_Y$ are dual and alternative to resp. $\rho_X$ and $\rho_Y$ then the convex programs for Eq. 4.8 and 4.9 are dual and alternative to the respective support functions. If (1) either $X$ or $Y$ is compact or (2) both $X$ and $Y$ are polyhedral, then also the convex program for Eq. 4.11 is dual and alternative to $\rho_{X\cap Y}$. If (3) $X \neq \emptyset$ and the convex program for $\bar{\rho}_X, \Lambda_X$ attains the infimum (when feasible), then the convex program for Eq. 4.10 is dual to $\rho_{\operatorname{coni} X}$.*

*Proof.* Let $Z$ be either $X+Y$, $AX$, $X\cap Y$, or $\operatorname{coni} X$. If $Z \neq \emptyset$ and $\rho_Z(a_Z) < +\infty$ then $d \neq 0$ and there exists a halfspace $H_Z = \{x \colon \langle a_Z, x\rangle \leq b_Z\}$ that strongly includes $X$. First, we show that there exists an assignment $H_X = \{x \colon \langle a_X, x\rangle \leq b_X\}$ and $H_Y = \{x \colon \langle a_Y, x\rangle \leq b_Y\}$ such that $H_X \subseteq X$ and $H_Y \supseteq Y$. For the operations $X+Y$, $AX$, and $X\cap Y$, by Thm. 3.1.3, 3.1.4, the hypotheses (1) and (2), Cor. 3.1.1.2, and Thm. 3.1.2, we have strong halfspace interpolation, namely there exist halfspace $H', H'' \neq \emptyset$ and that strongly include $X$ and $Y$. By the second part of Thm. 3.2.2, 3.2.3, and 3.2.1, the respective encoding admit assignments such that either $H_X = H'$ or $H_X = \mathbb{R}^n$ and either $H_Y = H''$ or $H_Y = \mathbb{R}^n$. For $\operatorname{coni} X$, the assiment exists by Lem. 4.4.1. Second, since $H_X \supseteq X$, then $\rho_X(a_X) \leq b_X < +\infty$ (note that, if $H_X = \mathbb{R}^n$ then $a_X = 0$ and $\rho_X(a_X) = 0$) and $\Lambda_X$ admits solution, since, by hypothesis, it satisfies duality; respectively, the same holds for $H_Y$. By the first parts of Thm. 3.2.2, 3.2.3, 3.2.1, and Lem. 4.4.1, if a solution exists and $X \subseteq H_X$ and $Y \subseteq H_Y$, then $Z \subseteq H_Z$, hence $\rho_Z(a_Z) < +\infty$. By the duality hypothesis, $X \subseteq H_X$ and $Y \subseteq H_Y$ hold for every solution and, hence, for every solution, $\rho_Z(a_X) < +\infty$. As a result, we have (i) $\rho_Z(a_Z) = +\infty$ if and only if a solution does not exists. Next, we show that, (ii) if a solution exists, then $\rho_Z(a_Z) = \inf\{\bar{\rho}_Z(a_Z) \colon \Lambda_Z(\lambda, a_Z)\}$, in other words, there exists a solution such that $\rho_Z(a_Z) + \varepsilon = \bar{\rho}_Z(\lambda, a_Z)$ and $\Lambda_Z(\lambda, a_Z)$, for every $\varepsilon > 0$. For $Z$ equal to every of $X + Y$, $AX$, and $X \cap Y$, strong halfspace interpolation guaratees that, for every $H_Z$ that includes $Z$ strongly, there exists a solution where $H_X$ and $H_Y$ resp. include $X$ and $Y$ strongly. Since, the programs for $\bar{\rho}_X$ and $\bar{\rho}_Y$ are also dual, for every $\varepsilon > 0$ a solution for $\rho_Z(a_Z) + \varepsilon|a_Z| = \bar{\rho}_Z(\lambda, a_Z)$ and $\Lambda_Z(\lambda, a_Z)$ exists. For $Z = \operatorname{coni} X$, by Lem. 4.4.1, for every $H_Z$ including $Z$ we have a halfspace $H_X$ including $X$, and, by hypothesis (3), there exists a solution $\rho_X(a_X) = \bar{\rho}_Z(\lambda, a_X) \leq b_X, \Lambda_X(\lambda, a_X)$. Finally, we show that the operations $X + Y$, $AX$, and $X \cap Y$ enjoy alternativity. Let $a_Z = 0$ and $b_Z = \epsilon < 0$ (i.e., $Z = \emptyset$). By strong halfspace interpolation, $H, H'$ that strongly include $X$ and $Y$ exist and, since $a_Z = 0$, by the second part of Thm. 3.2.2, 3.2.3, 3.2.1, also a solution such that either $H_X = H'$ or $H_X = \mathbb{R}^n$ (resp. for $H_Y$ and $H''$). If $H' \neq \emptyset$, then $\Lambda_X$ admits solution by the duality hypothesis, if $H' = \emptyset$, it admits solution by the alternativity hypothesis (resp. for $H_Y$ and $H''$). □

The construction allows us, not only to compute the support function, but also to compute sequences of halfspace interpolants. Then, we can extract the outward pointing directions by looking at the arguments instantiated by an emptiness check. For instance, $a_{X+Y}$ is the outward pointing direction of the halfspace containing $X+Y$ and, respectively, $a_X$ and $a_Y$ for the sets $X$ and $Y$. As a result, we can extract sequences of directions that refine the templates also arbitrary combinations of basic set operations, from one single emptiness check.

We build such a construction for arbitrary sequences of flow and jump operators induced by control paths. More concretely, let $w = v_0, e_1, v_1, \dots, e_k, v_k$ be a control path of some CHA $\mathcal{H}$ then the *path operator* of $w$ is

$$P_w = \mathrm{flow}_k \circ \mathrm{jump}_k \circ \cdots \circ \mathrm{flow}_1 \circ \mathrm{jump}_1 \circ \mathrm{flow}_0(Z_0). \tag{4.17}$$

By applying the above rules, we construct the convex program for the support function of $P_w$ as follows:

$$\begin{array}{lllr}
\text{minimize} & \bar\rho_{Z_0}(\lambda_{Z_0}) + \sum_{i=1}^k \bar\rho_{J_i}(\lambda_{J_i}) + \sum_{i=0}^k \bar\rho_{I_i}(\lambda_{I_i}) & & \\
\text{subject to} & (\lambda_{Z_0}, a_0 - b_0) \quad \in \Lambda_{Z_0}, & & \\
& (\lambda_{J_i}, \begin{bmatrix} -a_{i-1}, a_i - b_i \end{bmatrix}^\mathsf{T}) \quad \in \Lambda_{J_i} & \text{for each } i \in [1..k], & \\
& \bar\rho_{F_i}(\lambda_{F_i}, a_i - b_i) \quad \leq 0 & \text{for each } i \in [0..k], & (4.18) \\
& (\lambda_{F_i}, a_i - b_i) \quad \in \Lambda_{F_i} & \text{for each } i \in [0..k], & \\
& (\lambda_{I_i}, b_i) \quad \in \Lambda_{I_i} & \text{for each } i \in [0..k], & \\
& a_k \quad = d. & &
\end{array}$$

Duality and alternativity is preserved, therefore we can use such construction to compute the support functions for init and post (which are special cases of path).

**Lemma 4.4.3.** *For every CHA $\mathcal{H}$, every control path $w$ of $\mathcal{H}$, if either (1) all constraints are compact or (2) all constraints are polyhedral, if the convex programs for every constraint $X$ along the path are dual and alternative to $\rho_X$, and (3) the convex programs for $F_0, \dots, F_k$ attain the infimum (when feasible), then the convex program in Eq. 4.18 is dual and alternative to $\rho_{P_w}$.*

*Proof (sketch).* The convex program in Eq. 4.18 is the inlining of the construction of Eq. 4.8–4.11, for the operations composing $P_w$. To show this, we break down $\mathrm{flow}_v \circ \mathrm{jump}_e$ into six operations $X^{(1)} = [I_n \; 0_{n\times n}]^\mathsf{T} X^{(0)}$, $X^{(2)} = X^{(1)} + [0_{n\times n} \; I_n]^\mathsf{T}\mathbb{R}^n$, $X^{(3)} = X^{(2)} \cap J_e$, $X^{(4)} = [0_{n\times n} \; I_n] X^{(3)}$, $X^{(5)} = X^{(4)} + \mathrm{coni}\, F_v$, and $X^{(6)} = X^{(5)} \cap I_v$. Then, we use Eq. 3.18–3.20, and 4.12 to encode the interpolation problem over the directions $a_0, a_4, \dots, a_6 \in \mathbb{R}^n$, for, resp., $X^{(0)}, X^{(4)}, \dots, X^{(6)}$, $a_1, a_1', \dots, a_3, a_3' \in \mathbb{R}^n$, for, resp., $X^{(1)}, \dots, X^{(3)}$, $a_I, a_J, a_J', a_F \in \mathbb{R}^n$ for, resp., $I$, $J$, $F$ and $a_R, a_R' \in \mathbb{R}^n$ for the set $[0_{n\times n} \; I_n]^\mathsf{T}\mathbb{R}^n$. Note that unprimed and primed variables denote, resp., pre- and post-directions over the $2n$-dimensional constraints of the jump operator. We obtain the equations

$$
\begin{gathered}
a_6 = a_5 + a_I, a_5 = a_4 = a_F, a_4 = a_3', \\
\begin{pmatrix} 0 \cdot a_3 \\ a_3' \end{pmatrix} = \begin{pmatrix} a_2 \\ a_2' \end{pmatrix} + \begin{pmatrix} a_J \\ a_J' \end{pmatrix}, \begin{pmatrix} a_2 \\ a_2' \end{pmatrix} = \begin{pmatrix} a_1 \\ a_1' \end{pmatrix} = \begin{pmatrix} a_R \\ a_R' \end{pmatrix}, \begin{pmatrix} 0 \cdot a_R \\ a_R' \end{pmatrix} = 0, \\
a_1 = a_0.
\end{gathered}
\tag{4.19}
$$

Note that $(0 \cdot a_R, a_R') = 0$ corresponds to the constraint $\Lambda_{\mathbb{R}^n}(0 \cdot a_R, a_R')$. From the equations, we obtain the equalities $a_J = -a_0$ and $a_J' = a_F = a_0 - a_I$. We let $a_0 = a_{i-1}$, $a_6 = a_{i+1}$, $a_I = b_i$, and, after inlining all flow and jump operators, we obtain the convex program in Eq. 4.18. Then, by the hypotheses (1), (2), and (3), every operation satisfies the premises of Lem. 4.4.2. Then, by structural induction over the operations composing $P_w$, the convex program in Eq. 4.18 satisfies duality and alternativity w.r.t. $\rho_{P_w}$. □

We identify the arguments that determine a suitable sequence of halfspace interpolants after the emptiness check.

**Lemma 4.4.4.** *For every CHA $\mathcal{H}$, every control path $w$ of $\mathcal{H}$, every $\epsilon < 0$, and every $(\lambda^\star, 0) \in \Lambda_{P_w}$ whose projection on $a_0, a_1, \ldots, a_k$ is $a_0^\star, a_1^\star, \ldots, a_k^\star \in \mathbb{R}^n$, if the convex programs for the constraints $X$ along the path satisfy the conditions of Lem. 4.4.3 and $\bar{\rho}_{P_w}(\lambda^\star) \leq \epsilon$ then $a_0^\star, a_1^\star, \ldots, a_k^\star$ are the outward pointing directions of a sequence of halfspace interpolants $H_0, H_1, \ldots, H_k$ for $w$ as in Eq. 4.5.*

*Proof (sketch).* The variable $a_i$ respectively corresponds, w.r.t. the encoding of Sec. 3.2, to the variable $a_{\text{flow}_i \circ \text{jump}_i \circ \cdots \circ \text{flow}_1 \circ \text{jump}_1 \circ \text{flow}_0(Z_0)}$ and, hence, if $\bar{\rho}_{P_w}(\lambda^\star) < 0$, they constitute constitute the outward pointing directions of a sequence of interpolants for emptiness. □

In summary, we search by convex optimization for an argument for which the convex program of Eq. 6.20 for $d = 0$ has negative solution. If so, the argument $a_i^\star$ for the parameter $a_i$ is the outward pointing direction for the interpolant at mode $v_i$. Adding $a_i^\star$ to $\mathsf{prec}(v_i)$ eliminates the spurious counterexample $w$.

In this section, we have built a refiner for every spurious path of every CHA, assuming dual and alternative convex programs for the constraints along the path. In the following section, we discuss such functions and show how to instantiate interpolation for the special case of quadratic hybrid automata.

## 4.5 Abstraction Refinement for Quadratic Systems

The interpolation technique in Sec. 4.4 relies on the notions of duality and alternativity. Duality and alternativity are preserved by Minkowski sum, linear transformation, conical combination, and intersection, but whether they hold in the first place depends on the constraint of the automaton. We discuss these properties for (convex) quadratic programs, and we show their implications to the classes of quadratic and linear hybrid automata.

Closed convex quadratic sets are sets of the form $\bigcap_{i=1}^{m}\{x \in \mathbb{R}^n \mid xQ_ix^\mathsf{T} + p_i^\mathsf{T}x \leq r_i\}$ where $Q_1, \ldots, Q_m \in \mathbb{R}^{n\times n}$ are positive semidefinite matrices of coefficients, $p_1, \ldots, p_m \in \mathbb{R}^n$ are vectors of coefficients, and $r_1, \ldots, r_m \in \mathbb{R}$ are constants. Closed convex quadratic sets characterize quadratic hybrid automata.

**Definition 8** (Quadratic hybrid automata)**.** A quadratic hybrid automaton (QHA) is a CHA whose constraints define closed convex quadratic sets.

The support function of a convex quadratic set is a quadratically constrained (convex) quadratic program, which is known to cast to second-order conic programming (SOCP) [3]. We cast the support function to an optimization problem over a (rotated) second-order cone and we take its dual [3], so obtaining

$$\begin{aligned}
\text{minimize} \quad & r_1\lambda_1 + \cdots + r_m\lambda_m \\
\text{subject to} \quad & p_1\lambda_1 + L_1^\mathsf{T}\mu_1 + \cdots + p_m\lambda_m + L_m^\mathsf{T}\mu_m = d, \\
& \lambda_1 \geq \|\mu_1\|_2^2, \ldots, \lambda_m \geq \|\mu_m\|_2^2,
\end{aligned} \tag{4.20}$$

where $L_1, \ldots, L_m$ are the Cholesky decompositions of $Q_1, \ldots, Q_m$ respectively (i.e., $Q_i = L_iL_i^\mathsf{T}$), and $\lambda_1, \ldots, \lambda_m \in \mathbb{R}$ and $\mu_1, \ldots, \mu_m \in \mathbb{R}^n$ are the optimization arguments. Under the regularity conditions for non-linear optimization, e.g., Slater's condition, duality and alternativity hold [28; 3]. Encodings that do not need such conditions exist [96], but are not discussed in this paper.

Every algorithm that solves feasibility and optimization of SOCP solves init and post computation and halfspace interpolation for QHA, thus enabling their template-polyhedral abstraction and abstraction refinement.

**Theorem 4.5.1.** *Let $\mathcal{H}$ be a QHA with $n$ variables and $m$ inequalities, where either (i) all constraints are compact or (ii) all constraints are polyhedral. Let the time complexity of SOCP be* $\mathsf{socp}(\alpha, \beta, \gamma)$ *for $\alpha$ variables, $\beta$ equalities, and $\gamma$ cones.*

- *Init and post operators time complexity is* $p \times \mathsf{socp}(m, n, m)$ *where* $p = \max\{|\mathsf{prec}(v)| \mid v \in V\}$ *for the precision function* $\mathsf{prec}$.

- *Refinement time complexity is* $c \times \mathsf{socp}(m \times k, n \times k, m \times k)$ *where* $c = |W|$ *and* $k = \max\{|w| \mid w \in W\}$ *for the set of counterexamples* $W$.

*Proof (sketch).* Let $v_0, e_1, v_1, \dots, e_k, v_k$ be a control path of the QHA $\mathcal{H}$ and for all conditions $X$ along the path let "minimize $f_X\lambda_X$ subject to $A_X\lambda_X = d$, $\lambda_X \in C$" be the SOCP of Eq. 4.20 where $\lambda_X$,$f_X$, $A_X$, and $C_X$ are resp. optimization variable, objective function, equality constraint, and cones. Moreover, let's split $A_{J_e}$ in a upper $A'_{J_e}$ and a lower $A''_{J_e}$ of equal size. We instantiate the convex program in Eq. 4.18 with the SOCP if each constraint. We obtain

$$\begin{array}{lll}
\text{minimize} & f_{Z_0}\lambda_{Z_0} + \sum_{i=1}^{k} f_{J_i}\lambda_{J_i} + \sum_{i=0}^{k} f_{I_i}\lambda_{I_i} & \\
\text{subject to} & A_{Z_0}\lambda_{Z_0} + A_{I_0}\lambda_{I_0} = 0, & \\
& A''_{J_i}\lambda_{J_i} + A_{I_i}\lambda_{I_i} + A'_{J_{i+1}}\lambda_{J_{i+1}} = 0 & \text{for each } i \in [0..k-1], \\
& A_{F_i}\lambda_{F_i} + A_{I_i}\lambda_{I_i} + A'_{J_{i+1}}\lambda_{J_{i+1}} = 0 & \text{for each } i \in [0..k-1], \\
& A''_{J_k}\lambda_{J_k} + A_{I_k}\lambda_{I_k} = d, & \\
& A_{F_k}\lambda_{F_k} + A_{I_k}\lambda_{I_k} = d, & \\
& (\lambda_{Z_0}, \lambda_{F_0}, \lambda_{I_0}) \in C_{Z_0} \times C_{F_0} \times C_{I_0}, & \\
& (\lambda_{J_i}, \lambda_{F_i}, \lambda_{I_0}) \in C_{J_i} \times C_{F_i} \times C_{I_i} & \text{for each } i \in [1..k], \\
& f_{F_i}\lambda_{F_i} \leq 0 & \text{for each } i \in [0..k].
\end{array} \tag{4.21}$$

First, by Thm. 4.3.2 halfspace interpolants exist and solve the refinement problem, by Lem. 4.4.4, the convex program encodes the halfspace interpolation problem. Since $\mathcal{H}$ has $n$ variables, $m$ total (quadratic) inequalities, each SOCP in Eq. 4.20 has at most $n$ equalities (i.e., rows of $p$ and $L^\mathsf{T}$), $m$ variables and $m$ cones. Then, the SOCP in Eq. 4.21 has $m \times k$ variables, $n \times k$ linear equalities, and $m \times k$ cones. Init and post operators are special cases with $k = 1$. □

Nevertheless, the complexity SOCP remains an open problem on the Turing machine, while it is known to be in NP ∩ coNP on the real number model [96]. On the other hand, several efficient (but incomplete) numerical procedures are available, therefore in practice we can obtain support functions and interpolants, but with weaker guarantees. We are in a better position for the case of linear hybrid automata (LHA), i.e., the special case of

QHA where all constraints define polyhedra. For linear hybrid automata, the program of Eq. 4.20 is always a linear program (i.e., all cones are non-negativity constraints), where duality holds, alternativity is given by Farkas' lemma, and time complexity is polynomial. Hence, for LHA, init operator, post operator, and refinement time complexities are as well polynomial.

## 4.6 Benchmarks

We evaluate our algorithm on three main classes of benchmarks, namely Fischer's mutuals exclusion protocol [85], an adaptive cruise controller [73], and a synchronization protocol of the time-triggered Ethernet (TTEthernet) [25]. For each class, we consider a version with linear constraints and a version with nonlinear constraints, as well as for each a safe version, i.e. that does not reach the bad state, and an unsafe version, i.e. that reaches the bad state.

All of the benchmark consists of parallel compositions of hybrid automata, which differ from Def. 1 for the fact that they allow modeling of external and internal variables. Intuitively, external variables are variable whose name must refer to some variable of some other automaton in the composition, while internal variables are as for Def. 1. We do not formally define the composition rules here. For more details we refer the reader to the related work [87]. Note that such hybrid automata after parallel compositions are indeed hybrid automata as in Def. 1, hence our algorithm applies w.l.o.g.

In the following, we provide a detailed description of our benchmark classes and discuss some additional experimental results.

### 4.6.1 Fischer's mutual exclusion protocol

Fischer's protocol is a time based protocol of mutual exclusion between processes. It consists of m processes which coordinate using a m-dimensional vector of real variables $x$ (clocks) and a shared variable `k` which takes integer values between 0 and m.

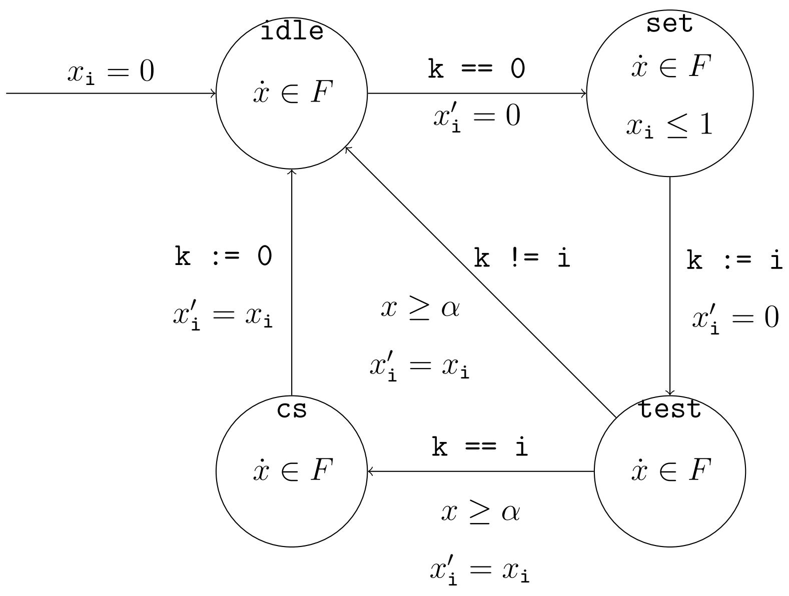
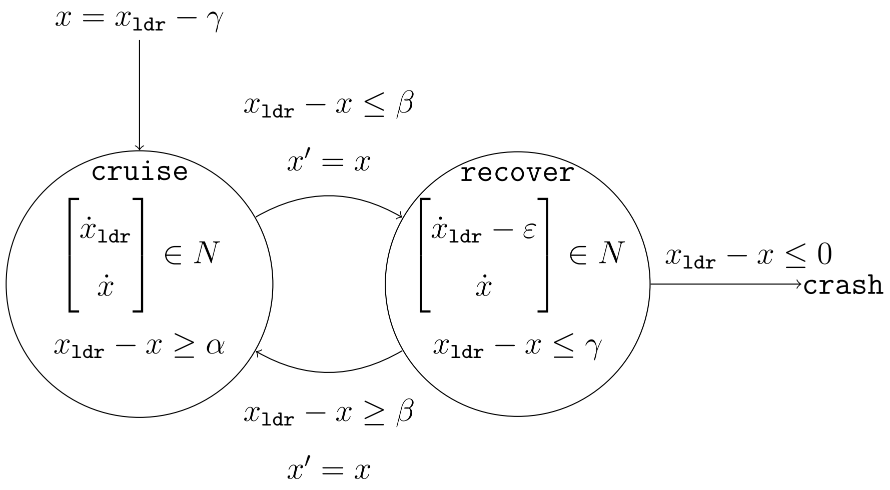


**Figure 4.4: Process automaton of the Fischer's protocol**

We model the shared variable as an automaton (without real-valued variables) with m+1 control modes. The variable automaton is in mode `i` exactly and only when variable `k` has value `i`. Each mode `i` has first an incoming control switch with event `k == i` from itself, second an incoming control switch with event `k != j` from itself, for each mode `j` $\neq$ `i`, and third an incoming control switch with event `k := j` from each mode `j`.

The process automaton is depicted in Fig. 4.4. Each process is an instance of this automaton and has an associated index `i` between 1 and m and an associated clock $x_{\mathtt{i}}$, which is an element of $x$. The automaton consists of the modes `idle`, `set`, `test`, and `cs`. Each process starts in mode `idle`. When process `i` wants to enter the critical section `cs` it first resets $x_{\mathtt{i}}$ to 0 (entering `set`) and then it sets `k` to `i` (entering `test`) before the clock $x_{\mathtt{i}}$ hits value 1. Afterwards (while entering `test`) it resets $x$ to 0 again and then it tests whether `k` equals `i` only after the clock $x$ hits $\alpha$. If `k` equals `i` then the process enters the critical section `cs`, otherwise it repeats from `idle`. The process always resets `k` to `0` while exiting the critical section.

The shared variable automaton and the process automata synchronize on all events `k := 0`, …, `k := m`, `k == 0`, …, `k == m`, and `k != 1`, …, `k != m`. The variable $x_{\mathtt{i}}$ is internal and all variables $x_{\mathtt{j}}$, for `j` $\neq$ `i`, are external to process `i`.



$x = x_{\mathtt{ldr}} - \gamma$
cruise
$\begin{bmatrix} \dot{x}_{\mathtt{ldr}} \\ \dot{x} \end{bmatrix} \in N$
$x_{\mathtt{ldr}} - x \geq \alpha$
$x_{\mathtt{ldr}} - x \leq \beta$
$x' = x$
recover
$\begin{bmatrix} \dot{x}_{\mathtt{ldr}} - \varepsilon \\ \dot{x} \end{bmatrix} \in N$
$x_{\mathtt{ldr}} - x \leq \gamma$
$x_{\mathtt{ldr}} - x \geq \beta$
$x' = x$
$x_{\mathtt{ldr}} - x \leq 0$
crash

**Figure 4.5: Follower automaton of the adaptive cruise controller**

The protocol is correct if two processes are never in the critical section at the same time. Hence, the question is whether a state with two protocols in `cs` is reachable. The correctness depends on the relation between the *clock drift* $F \subseteq \mathbb{R}^m$ and $\alpha \in \mathbb{R}$. Different versions of $F$ and $\alpha$ define exactly the different versions of the protocol we are considering. In the linear case, $F$ is the m-dimensional unit cube centered in 1, i.e., the constraint over dotted variables given by $\frac{1}{2} \leq \dot{x}_1 \leq \frac{3}{2}, \ldots, \frac{1}{2} \leq \dot{x}_{\mathtt{m}} \leq \frac{3}{2}$, while $\alpha$ equals 2.1 for the safe case and 1.9 for the unsafe case. In the non-linear case, $F$ is the m-dimensional unit ball centered in 1, i.e., the constraint over dotted variables given by $\sqrt{\dot{x}_1^2 + \ldots, \dot{x}_1^2} \leq 1$, while $\alpha$ equals 1.42 for the safe case and 1.40 for unsafe case (over- and under-approximations of $\sqrt{2}$ respectively). We verify the linear cases up to 5 processes and the non-linear cases up to 3 processes..

### 4.6.2 Adaptive cruise controller

The adaptive cruise controller is a distributed system of distance control of a platoon of cars on a straight highway. It consists of a sequence of m cars each of which is called the *follower* of the previous car in the sequence and the *leader* of the next car in the sequence. The task of each follower is to control the speed to the speed of the leader and keeping a safety distance. The first car in the sequence drives at a constant speed.

The automaton for the first car, the leader automaton, consists of one single mode `cruise` and one single internal variable $x$. The variable $x$ models the position of the car and is initialized at value $m\gamma$ and its flow consists of a constant positive real value.

The automaton for all other cars, the follower automaton, is shown in Fig. 4.5. It consists of the modes `cruise`, `recover`, and `crash`, one external variable $x_{\mathtt{ldr}}$ and one internal variable $x$. Variable $x$ models the position of this car and variable $x_{\mathtt{ldr}}$ models the position of its leader. The flow $N$ models the *speed measurement drift* between follower and leader, e.g., due to sensing noise and other environmental conditions. The switching logic is governed by the constants $0 < \alpha < \beta < \gamma$. The car starts in mode `cruise` with distance $\gamma$ from its leader. When the distance falls between $\beta$ and $\alpha$, a switch to `recover` happens. In `recover` the follower slows down of a constant $\epsilon > 0$ w.r.t. the speed of the leader. When the distance establishes between $\beta$ and $\gamma$, a switch back to `cruise` happens. Otherwise, if the distance keeps falling and hits 0, the car switches to `crash`. We omit the constraints of `crash` as they are immaterial for our purpose.

The whole model consists of the parallel composition of one leader and m-1 followers. The variable $x_{\mathtt{ldr}}$ of each follower is renamed after the variable $x$ of its leader. No event synchronizes.

We check for the reachability of any mode in which at least one car is in mode `crash`. The distance thresholds $\alpha$, $\beta$, and $\gamma$ are constants and do not affect the reachability of `crash`, but rather the smoothness of the system (switching frequency), therefore we fix them to 1, 2, and 3 respectively. Reachability if affected by $N$ and $\varepsilon$. For the linear case we define them as follows. The flow $N$ is a two-dimensional unit cube centered in 0, namely it defines flow $|\dot{x} - \dot{x}_{\mathtt{ldr}}| \leq \frac{1}{2}$ in `cruise` and $|\dot{x} - \dot{x}_{\mathtt{ldr}} + \varepsilon| \leq \frac{1}{2}$ in `recovery`. The constant $\varepsilon$ is 1.1 for the safe case and 0.9 for the unsafe case. For the non-linear case we replace $N$ with a unit ball centered in 0, and $\varepsilon$ with 1.42 for the safe case and 1.40 for the unsafe case. We analyze platoons up to 7 cars.

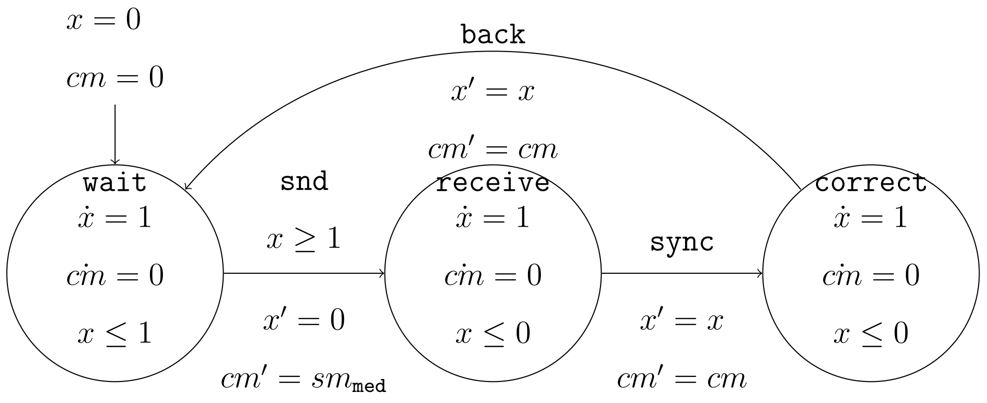


**Figure 4.6: Compression master of the TTEthernet protocol**

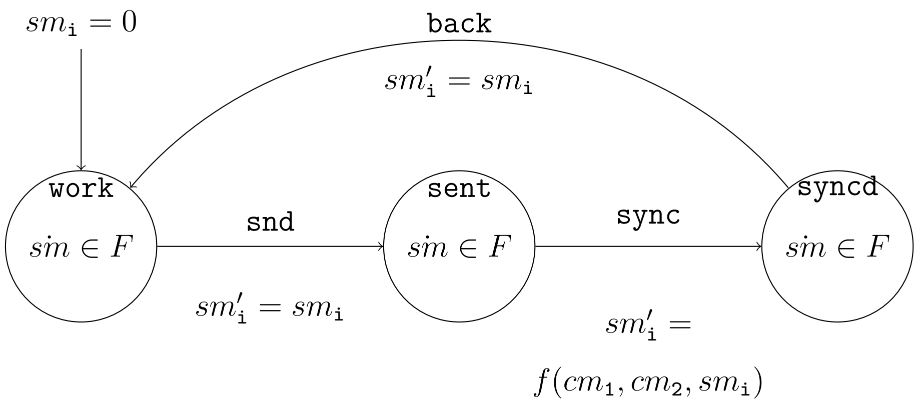


**Figure 4.7: Synchronization master of the TTEthernet protocol**

### 4.6.3 Time-triggered Ethernet synchronization protocol

We describe the protocol for the remote synchronization of possibly drifted clocks in a TTEthernet. It consists of a set of two *compression masters* and $m$ *synchronization masters*, all sharing a channel. First, the synchronization masters send their clock values to all compression masters. Then compression masters choose the median among all clocks. Second, the two compression masters send the median to all synchronization masters. Then all synchronization masters update their own clocks with the average of the medians.

The automaton for the compression master is shown in Fig. 4.6. It consists of three modes `wait`, `receive`, and `correct`, two internal variables $x$ and $cm$ and one external variable $sm_{\mathtt{med}}$. The variable $x$ is the internal clock. The automaton starts in `wait`. The protocol begin when the internal clock $x$ hits exactly 1 synchronizing on event `snd`. It stores in $cm$ the median of the synchronization master clocks $sm_{\mathtt{med}}$ and switches to `receive`. Now the automaton immediately (within zero time) performs a `sync` event in which it sends its $cm$ value to the synchronization masters and switches to `correct`. Again the automaton immediately performs a `back` event and switches back to `wait`.

The automaton for the synchronization master is shown in Fig. 4.7. It consists of three modes `work`, `sent`, and `syncd`, one $m$ dimensional variable $sm$ and two variables $cm_1$ and $cm_2$. One element of $sm$ is internal, denoted $sm_{\mathtt{i}}$, while all others are external. They model internal clock and clocks of the other synchronization masters, respectively. Both $cm_1$ and $cm_2$ are external. The clocks $sm$ have possibly drifted speed according to the flow $F$. The automaton starts in `work` and eventually sends its internal clock to the compression masters on event `snd`, switching to `sent`. Afterward it receives the values of the two compression masters $cm_1$ and $cm_2$. If computes the function value $f(cm_1, cm_2, sm_{\mathtt{i}})$ and uses it to update its own clock on event `sync`. It switches to `syncd` and then back to `work`.

We compose in parallel the two compression masters and the $m$ synchronization masters. We arbitrarily choose one of the synchronization master clocks as median value (as hybrid automata are inherently non-deterministic, there is not a probabilistic distribution among clocks drifts) and we rename $sm_{\mathtt{med}}$ of both compression masters to it (similarly to the related literature [25]). We rename $cm_1$ and $cm_2$ of each synchronization master to the variable $cm$ of the two compression masters. The automata synchronize on all events.

The protocol aims at guaranteeing a bounded clock difference between the clocks of any two synchronization masters. Let us denote the bound with $\alpha$. We compose the systems with a monitor that start in some mode `good` and switches to some mode `bad` as soon as the difference between any pair of $sm$ falls above $\alpha$. The reachability question is weather `bad` is reachable. The answer depends on $F$, $\alpha$, and $f$. Again, we distinguish between a linear and a non linear case. For the linear case $F$ is the cube of side 2 centered in 1, i.e., the flow $0 \leq \dot{sm}_{\mathtt{1}} \leq 1, \ldots, 0 \leq \dot{sm}_{\mathtt{m}} \leq 1$, and $\alpha$ is 2.1. For the non-linear case $F$ is the unit ball centered in 1, i.e., the flow $\sqrt{(\dot{sm}_{\mathtt{1}} - 1)^2 + \ldots (\dot{sm}_{\mathtt{m}} - 1)^2} \leq 1$, and $\alpha$ is 1.42. In both linear and non-linear case, for the safe case the update function $f$ is equal to the average between the two compression master values $\frac{cm_1 + cm_2}{2}$, for the unsafe case it is equal to $sm_{\mathtt{i}}$. We verify both linear and non-linear systems with 3,5,9, and 17 synchronization masters.

## 4.7 Experimental Evaluation

We evaluate our algorithms on three main classes of benchmarks, namely Fischer's protocol [85], an adaptive cruise controller [73], and the TTEthernet protocol [25]. For each class, we consider a linear version and a non-linear version, as well as for each a safe version and an unsafe version.

Fischer's protocol is a time based protocol of mutual exclusion between processes. The protocol is correct if two processes are never in the critical section at the same time. For the linear version, the flow constraints are given by $\frac{1}{2} \leq \dot{x}_{\mathtt{1}} \leq \frac{3}{2}, \ldots, \frac{1}{2} \leq \dot{x}_{\mathtt{m}} \leq \frac{3}{2}$, where $x_{\mathtt{i}}$ is the clock of the `i`-th process, and for the non-linear case, $\sqrt{\dot{x}_{\mathtt{1}}^2 + \cdots + \dot{x}_{\mathtt{m}}^2} \leq 1$. We verify the linear version up to 5 processes and the non-linear version up to 3 processes.

The adaptive cruise controller is a distributed system for safety distance of platoon of cars. Each car either cruises or recovers by slowing down. The relative velocity has a drift $|\dot{x} - \dot{x}_{\mathtt{ldr}}| \leq \frac{1}{2}$ when cruising and $|\dot{x} - \dot{x}_{\mathtt{ldr}} + \varepsilon| \leq \frac{1}{2}$ when recovering, where $x$ and $x_{\mathtt{ldr}}$ are the positions of each car the car in front, resp, and $\varepsilon$ is the slow-down. We check for car crashes in platoons up to 7 cars.

Finally, we consider the TTEthernet protocol for the remote synchronization of possibly drifted clocks distributed over multiple components. Similarly to previous case studies, we consider flows defined in terms of intervals and unit balls for linear and non-linear cases, respectively. We verify both linear and non-linear systems with 3, 5, 9, and 17 components.

| Benchmark | Empty | | | | | Octagonal | | | | | PHAVer |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | #spu | #dir | cgr [s] | itp [s] | ver [s] | #spu | #dir | cgr [s] | itp [s] | ver [s] | time [s] |
| `fsr_lnr_2_sf` | 5 | 8 | 0.06 | 0.02 | ≈0 | 0 | 256 + 0 | 0.11 | 0 | 0.11 | ≈0 |
| `fsr_lnr_3_sf` | 41 | 69 | 1.12 | 0.02 | 0.02 | 12 | 3456 + 12 | 5.55 | ≈0 | 0.50 | 1.25 |
| `fsr_lnr_4_sf` | 259 | 440 | 33.16 | 0.29 | 0.14 | 221 | 32768 + 221 | 1190 | 0.07 | 23.06 | 135 |
| `fsr_lnr_5_sf` | 1379 | 2335 | 857 | 2.08 | 0.76 | oot | 256k | oot | oot | oot | 78807 |
| `fsr_lnr_2_usf` | 0 | 0 | ≈0 | 0 | ≈0 | 0 | 256 + 0 | 0 | 0 | 0.12 | ≈0 |
| `fsr_lnr_3_usf` | 0 | 0 | 0.03 | 0 | 0.03 | 0 | 3456 + 0 | 0 | 0 | 0.37 | 1.01 |
| `fsr_lnr_4_usf` | 0 | 0 | 0.06 | 0 | 0.06 | 0 | 32768 + 0 | 0 | 0 | 1.67 | 300 |
| `fsr_lnr_5_usf` | 0 | 0 | 0.16 | 0 | 0.16 | 0 | 256k + 0 | 0 | 0 | 13.63 | oom |
| `fsr_qdr_2_sf` | 5 | 8 | 5.13 | 0.10 | 1.32 | 0 | 256 + 0 | 0 | 0 | 8.18 | - |
| `fsr_qdr_3_sf` | 41 | 69 | 226 | 0.44 | 9.04 | 12 | 3456 + 12 | 3599 | 0.15 | 886 | - |
| `fsr_qdr_2_usf` | 0 | 0 | 0.66 | 0 | 0.66 | 0 | 256 + 0 | 0 | 0 | 6.40 | - |
| `fsr_qdr_3_usf` | 0 | 0 | 1.76 | 0 | 1.76 | 0 | 3456 + 0 | 0 | 0 | 26.67 | - |
| `acc_lnr_2_sf` | 2 | 2 | ≈0 | ≈0 | ≈0 | 0 | 32 + 0 | 0 | 0 | ≈0 | ≈0 |
| `acc_lnr_3_sf` | 8 | 8 | 0.04 | ≈0 | ≈0 | 0 | 144 + 0 | 0 | 0 | 0.19 | 0.03 |
| `acc_lnr_4_sf` | 24 | 24 | 0.39 | ≈0 | 0.02 | 0 | 512 + 0 | 0 | 0 | 0.87 | 0.53 |
| `acc_lnr_5_sf` | 64 | 64 | 0.94 | ≈0 | 0.12 | oot | 1600 | oot | oot | oot | 21.78 |
| `acc_lnr_6_sf` | 160 | 160 | 42.12 | 0.07 | 0.74 | oot | 4608 | oot | oot | oot | 1455 |
| `acc_lnr_7_sf` | 384 | 384 | 569 | 0.13 | 4.22 | oot | 12544 | oot | oot | oot | oot |
| `acc_lnr_2_usf` | 1 | 1 | ≈0 | ≈0 | ≈0 | 0 | 32 + 0 | 0 | 0 | ≈0 | ≈0 |
| `acc_lnr_3_usf` | 2 | 2 | ≈0 | ≈0 | ≈0 | 0 | 144 + 0 | 0 | 0 | 0.05 | ≈0 |
| `acc_lnr_4_usf` | 3 | 3 | ≈0 | ≈0 | ≈0 | 0 | 512 + 0 | 0 | 0 | 0.37 | 0.18 |
| `acc_lnr_5_usf` | 4 | 4 | ≈0 | ≈0 | ≈0 | 0 | 1600 + 0 | 0 | 0 | 0.61 | 22.51 |
| `acc_lnr_6_usf` | 5 | 5 | 0.06 | ≈0 | 0.04 | 0 | 4608 + 0 | 0 | 0 | 1.23 | 4621 |
| `acc_lnr_7_usf` | 6 | 6 | 0.17 | ≈0 | 0.06 | 0 | 12544 + 0 | 0 | 0 | 2.87 | oot |
| `tte_lnr_3_sf` | 17 | 18 | 0.17 | ≈0 | ≈0 | oot | 864 | oot | oot | oot | oot |
| `tte_lnr_5_sf` | 49 | 50 | 0.32 | ≈0 | ≈0 | oot | 2400 | oot | oot | oot | oot |
| `tte_lnr_9_sf` | 161 | 162 | 3.47 | ≈0 | 0.06 | oot | 7776 | oot | oot | oot | oot |
| `tte_lnr_17_sf` | 577 | 578 | 239 | 0.06 | 1.27 | oot | 27774 | oot | oot | oot | oot |
| `tte_lnr_3_usf` | 18 | 24 | 0.26 | ≈0 | 0.05 | 0 | 864 + 0 | 0 | 0 | 0.42 | oot |
| `tte_lnr_5_usf` | 60 | 80 | 0.85 | ≈0 | 0.02 | 0 | 2400 + 0 | 0 | 0 | 0.95 | oot |
| `tte_lnr_9_usf` | 216 | 288 | 15.65 | ≈0 | 0.26 | 0 | 7776 + 0 | 0 | 0 | 4.36 | oot |
| `tte_lnr_17_usf` | 816 | 1088 | 1722 | 0.35 | 8.68 | 0 | 27774 + 0 | 0 | 0 | 109 | oot |
| `tte_qdr_3_sf` | 17 | 18 | 8.30 | 0.38 | 1.36 | oot | 864 | oot | oot | oot | - |
| `tte_qdr_5_sf` | 49 | 50 | 56.31 | 1.25 | 4.01 | oot | 2400 | oot | oot | oot | - |
| `tte_qdr_9_sf` | 161 | 162 | 492 | 3.94 | 12.29 | oot | 7776 | oot | oot | oot | - |
| `tte_qdr_17_sf` | 577 | 578 | 3325 | 12.79 | 47.49 | oot | 27774 | oot | oot | oot | - |
| `tte_qdr_3_usf` | 18 | 24 | 3.65 | 0.21 | 0.60 | 0 | 864 + 0 | 0 | 0 | 6.33 | - |
| `tte_qdr_5_usf` | 60 | 80 | 37.99 | 0.66 | 1.82 | 0 | 2400 + 0 | 0 | 0 | 21.68 | - |
| `tte_qdr_9_usf` | 216 | 288 | 514 | 2.61 | 7.32 | 0 | 7776 + 0 | 0 | 0 | 58.27 | - |
| `tte_qdr_17_usf` | 816 | 1088 | 15515 | 18.28 | 58.95 | 0 | 27774 + 0 | 0 | 0 | 78.19 | - |

**Table 4.1: Runtimes for the reachability analysis of Fishers' protocol, adaptive cruise controller, and TTEthernet.**

We implemented a CEGAR loop based on our procedure in C++ and conducted the following experiments on a machine with 2.6GHz CPU and 4 GB of dedicated RAM. We use the GLPK for solving LPs and MOSEK for solving SOCPs [1; 90]. We executed our tool under the *empty* strategy and the *octagonal* strategy. With the empty strategy, the initial precision is empty, which means that the very first abstraction computation consists of a simple exploration of the control graph. With the octagonal strategy, the precision at every mode consists of the octagonal template, with a total of $2|V|n^2$ directions over all modes. For all linear instances, we compared against PHAVer [52] (SpaceEx v0.9.8c with PHAVer scenario).

Table 4.1 shows the results. Specifically, empty and octagonal indicate the initial precision; #spu is number of discovered spurious counterexamples, #dir is the number of discovered directions (empty case) or initial directions + discovered directions (octagonal case); cgr is the total time spent in unsuccessful abstractions (with spurious counterexample), itp is the total time spent in discovering halfspace interpolants, ver is the time spent in successful abstractions; oot indicates out of time (24 hours), oom indicates out of memory (4Gb), and dash indicates unsupported. The benchmark names are structured as follows. `fsr` indicates Fischer's protocol, `acc` indicates adaptive cruise controller, `tte` indicates TTEthernet, `lnr` indicates linear, `qdr` indicates quadratic, the following number indicates the number of components, and `sf` and `usf` resp. indicate safe and unsafe.

The empty strategy has on average the best runtime and always outperforms PHAVer. It also outperforms the octagonal strategy for most of the instances. Both strategies spend most of the time in the first phase (CEGAR iterations ending in a spurious counterexample), and take a very short time for the final verification step. For Fischer's protocol the octagonal strategy is always slower than the empty. For the other benchmarks the difference is less stunning, in particular for the unsafe cases of the TTEthernet benchmarks, where the first phase penalizes considerably. On the other hand, we can observe that, under the assumption that we are not aware of the safety of the systems, our method shows to be the most scalable. The octagonal strategy tends to run out of time because the higher number of directions causes the generation of bigger and bigger abstract regions. In fact, we have verified that for these instances a spurious counter-example is never found. The same argument likely holds for PHAVer, as its dump shows that new symbolic states are always found. Not surprisingly, for QHA the performance is generally worse than for LHA.

In summary, template polyhedra coupled with our abstraction refinement technique are faster than the exact polyhedral reachability analysis. Noteworthy is how negligible is the time required in the final verification step on all instances. Our tool recomputes the whole abstraction after every refinement phase, as all our efforts have been strictly focused on implementing an efficient template refinement. The final time sets a lower bound for the verification time achievable by an incremental abstraction. Furthermore, we could observe that inferring small template sets plays an important role in the convergence of the whole analysis.

# 5 Conic Abstractions for Affine Hybrid Automata

For the purpose of unbounded-time analysis, a very useful strategy is to use lightweight runtime technique for continuous online verification[74; 75], and another important strategy is to abstract the original system and find an invariant for the abstraction [68]. However, obtaining a high-quality abstraction automatically for the original system is challenging by itself and this is why *PHAVer* chooses to leave this important work to users, who have some domain expertise available for this purpose [52]. Roughly speaking, the ultimate goal of abstraction is to use a partition of the state space which is as coarse as possible, to derive an over-approximation of the original system which is as accurate as possible and allows a computation of the reachable state set which is as efficient as possible. Depending on the set representation that is used, the schemes that have been proposed for state space partition vary significantly [19; 78; 9; 110; 109; 101; 106; 14; 65]. When polyhedra are used for the set representation of states, a guiding principle for state space partitioning is that the partition should result in a set of regions that are as "straight" as possible. By "straight region", we mean that the maximal angle between the derivative vectors in that region (which we define as the *twisting* of the region) is small, so that every trajectory tends to be straight in the region. The benefit of straight regions is that they can be over-approximated accurately by polyhedra. However, for a given system, obtaining the least number of straight regions under a given threshold of twisting is by no means trivial.

With this principle in mind, we propose a new abstraction called *conic abstraction* for affine hybrid systems and we compute reachable state sets based on the abstraction. Given an $n$-D linear system defined by $\dot{\vec{x}} = \boldsymbol{A}\vec{x}$, assume that $\boldsymbol{A}$ is an invertible matrix (note that any affine system $\dot{\vec{x}} = \boldsymbol{A}\vec{x} + \vec{b}$ can be transformed into a linear system under this assumption). The basic idea behind conic abstraction is as follows. First, the derivative space of the system is partitioned uniformly into a set $\boldsymbol{D}$ of convex polyhedral cones. Then, $\boldsymbol{D}$ is mapped back from the derivative space to the state space to obtain a conic partition $\boldsymbol{C}$ of state space, i.e., $\forall C_i \in \boldsymbol{C} : \exists D_i \in \boldsymbol{D} : C_i = \{\boldsymbol{A}^{-1}\vec{y} \mid \vec{y} \in D_i\}$. Finally, every state region $C_i$ is treated as a discrete location ("mode") and the discrete transitions between these modes are decided on-the-fly according to whether there exists a trajectory between them. By doing so, we can easily obtain the differential inclusion $D_i$ for each polyhedral cone $C_i$. Therefore, for any subset $I_i$ of $C_i$, the reachable set of $I_i$ in $C_i$ can be overapproximated by $(I_i \oplus D_i) \cap C_i$, where $\oplus$ denotes the Minkowski sum. More importantly, since the twisting of $C_i$ is determined by the maximal angle of $D_i$, the partition can be refined easily to any desired precision, by shrinking the maximal angle of the conic partition of the derivative space. Note that an important feature of $C_i$ is that it is an unbounded set, however, with a bounded twisting, which means that each $C_i$ captures infinitely long trajectories only if they are straight enough. *Diagonalizable* affine systems, for which the matrix $\boldsymbol{A}$ is diagonalizable, form such a class of systems, because for diagonalizable systems all trajectories eventually evolve into approximately straight lines.

Using properties of diagonalizable affine systems, we develop an algorithm that constructs a conic abstraction as a directed acyclic graph (DAG) for which an invariant (i.e., an over-approximation of the reachable state set) exists and the computation of the invariant is guaranteed to terminate. The algorithm is implemented in a tool and experiments on randomly generated examples as well as published benchmarks show that our approach is more powerful than *PHAVer* in finding unbounded invariants. Note that computing an unbounded invariant for diagonalizable affine systems lies beyond the capability of tools for time-bounded reachability analysis, such as *SpaceEx* [57].

The main contributions of this paper are as follows. First, we propose conic abstractions and a method for constructing them for affine hybrid systems. The core idea lies in deriving a state space partition from a uniform partition of the derivative space. Second, we develop an algorithm for building conic abstractions as DAGs for diagonalizable affine systems and for computing invariants on these abstractions. Finally, we implement and evaluate our approach in a tool.

The paper is organized as follows. Section 5.1 is devoted to preliminary definitions. In Section 5.2, we introduce conic abstractions for affine systems. In Section 5.3, we show how to construct conic abstractions as DAGs for diagonalizable systems. Section 5.4 describes how we compute invariants for continuous systems and affine hybrid systems. In Section 5.5, we present our experimental results.

## 5.1 Affine Systems

In this section, we recall some concepts used throughout the paper. We first clarify some notation conventions. We use bold uppercase letters such as $\boldsymbol{A}$ to denote matrices and bold lowercase letters such as $\vec{b}$ to denote vectors and $diag(\lambda_1, \cdots, \lambda_n)$ to denote a diagonal matrix with $\lambda_1, \cdots, \lambda_n$ as its diagonal elements. We call a dynamical system defined as $\dot{\vec{x}} = \boldsymbol{A}\vec{x} + \vec{b}$ an affine system and we use a superscript $T$ for the transpose of a matrix.

**Definition 9** (Affine System)**.** An $n$-dimensional affine system consists of a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and a vector $\vec{b} \in \mathbb{R}^n$, which define the vector flow $\dot{\vec{x}} = \boldsymbol{A}\vec{x} + \vec{b}$, and an initial region $X_0 \subseteq \mathbb{R}^n$ defined by a polyhedron.

Whenever the initial set is immaterial, we refer to an affine system just as to $\dot{\vec{x}} = \boldsymbol{A}\vec{x} + \vec{b}$. We next introduce the concept of Lie derivative.

**Definition 10** (Lie derivative)**.** For a given polynomial $p \in \mathbb{K}[\vec{x}]$ and a continuous system $\dot{\vec{x}} = \vec{f}$, the **Lie derivative** of $p \in \mathbb{K}[\vec{x}]$ along $\vec{f}$ is defined as $\mathcal{L}_{\vec{f}}p \overset{\text{def}}{=} \langle \nabla p, \vec{f}^T \rangle$.

For an affine system $\dot{\vec{x}} = \boldsymbol{A}\vec{x} + \vec{b}$, we can simply write the Lie derivative as $\mathcal{L}_{\vec{\boldsymbol{A}}}\langle \vec{a}, \vec{x}\rangle = \langle \vec{a}\boldsymbol{A}, \vec{x}^T\rangle + \langle \vec{a}, \vec{b}^T\rangle$. We call a polyhedral cone $C$ an intersection of linear inequalities of the form $\langle \vec{a}, \vec{x}\rangle \leq 0$, and we denote its boundary as $\partial C$. For $X, Y \subseteq \mathbb{R}^n$, $X \oplus Y$ denotes their Minkowski sum $\{\vec{x} + \vec{y} \colon \vec{x} \in X \text{ and } \vec{y} \in Y\}$, and for $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $X \subseteq \mathbb{R}^n$, $\boldsymbol{A}X$ denotes the linear transformation $\{\boldsymbol{A}\vec{x} \colon \vec{x} \in X\}$.

## 5.2 Conic Abstractions of Affine Systems

Discrete abstraction is a basic strategy for verifying continuous and hybrid systems. There are many abstraction approaches proposed for this purpose. Rectangular abstraction [68; 78; 52] and nonlinear abstraction [9; 110; 109; 106] are widely used. However, even for linear systems, the existing abstraction approaches are still inefficient. In this section, focusing on linear systems, we propose a new abstraction approach called *conic abstraction*. However, since every affine system can be transformed into an equivalent linear system $\dot{\vec{x}} = \boldsymbol{A}\vec{x}$, as we discuss in Sec. 5.3, our discussion applies to affine systems too.

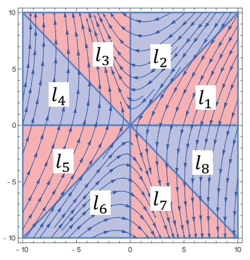
The idea is that we partition the state space of a linear system into a set of convex polyhedral cones. We call this set a *conic partition*.

**Definition 11** (Conic Partition)**.** A conic partition is a set of polyhedral cones $\Delta$ such that $\cup_{C_i \in \Delta} C_i = \mathbb{R}^n$ and every two cones $C_1, C_2 \in \Delta$ have disjoint interiors, i.e., $(C_1 \backslash \partial C_1) \cap (C_2 \backslash \partial C_2) = \emptyset$.

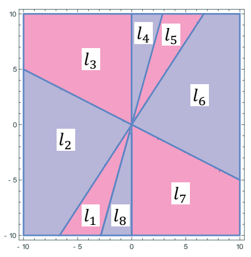
We call an element of the partition $C \in \Delta$ a region. Then we construct a graph whose vertices correspond to partition regions and edges indicate possible flow between them. We call such a graph a *conic abstraction*.

**Definition 12** (Conic Abstraction)**.** The conic abstraction of the linear system $\dot{\vec{x}} = \boldsymbol{A}\vec{x}$ derived from the conic partition $\Delta$ consists of the finite directed graph $(L, E)$ as follows. Every vertex $l_C \in L$ corresponds to one and only one cone $C \in \Delta$. There exists an edge $(l_{C_1}, l_{C_2}) \in E$ if and only if there exists a plane $F_1 = \{\vec{x} \mid \langle \vec{a}, \vec{x}^T \rangle = 0\}$ such that 1) $\partial C_1 \cap \partial C_2 \subseteq F_1$, 2) $C_1 \subseteq \{\vec{x} \mid \langle \vec{a}, \vec{x}^T \rangle \leq 0\}$, 3) the Lie derivative of $\langle \vec{a}, \vec{x}^T \rangle$ is non-negative at some common point, i.e., $\mathcal{L}_{\vec{\boldsymbol{A}}} \langle \vec{a}, \vec{x}^T \rangle \geq 0$ for some $\vec{x} \in \partial C_1 \cap \partial C_2$.

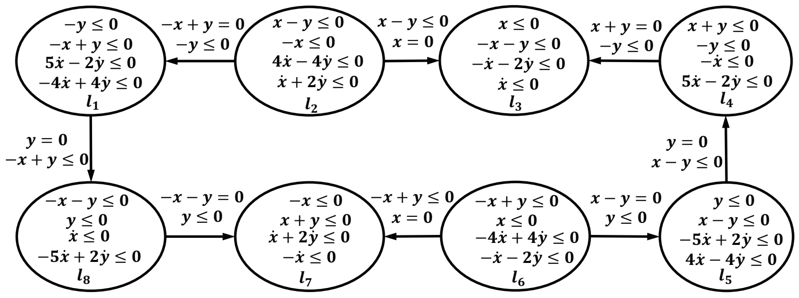
We elaborate on how to construct a conic abstraction for diagonalizable systems in Sec. 5.3. A conic abstraction can be seen as a Linear Hybrid Automaton (LHA, [63]), whose locations $l_C$ are such that its invariant is given by $C$, its flow is given by a differential inclusion defined as $\dot{\vec{x}} \in \boldsymbol{A}C$, and whose switch guards consist of the common facet of the respective adjacent cones. Consider the linear system described by $\dot{x} = -2x - 2y, \dot{y} = -5x + y$. A conic partition of the state space, the corresponding differential inclusion and the conic abstraction of the system is shown in Figure 5.1a, Figure 5.1b and Figure 5.1c, respectively. As you can see, both the invariant and the



(a)



(b)



$-y \le 0$, $-x + y \le 0$, $5\dot{x} - 2\dot{y} \le 0$, $-4\dot{x} + 4\dot{y} \le 0$, $l_1$
$-x + y = 0$, $-y \le 0$
$x - y \le 0$, $-x \le 0$, $4\dot{x} - 4\dot{y} \le 0$, $\dot{x} + 2\dot{y} \le 0$, $l_2$
$x - y \le 0$, $x = 0$
$x \le 0$, $-x - y \le 0$, $-\dot{x} - 2\dot{y} \le 0$, $\dot{x} \le 0$, $l_3$
$x + y = 0$, $-y \le 0$
$x + y \le 0$, $-y \le 0$, $-\dot{x} \le 0$, $5\dot{x} - 2\dot{y} \le 0$, $l_4$
$y = 0$, $-x + y \le 0$
$y = 0$, $x - y \le 0$
$-x - y \le 0$, $y \le 0$, $\dot{x} \le 0$, $-5\dot{x} + 2\dot{y} \le 0$, $l_8$
$-x - y = 0$, $y \le 0$
$-x \le 0$, $x + y \le 0$, $\dot{x} + 2\dot{y} \le 0$, $-\dot{x} \le 0$, $l_7$
$-x + y \le 0$, $x = 0$
$-x + y \le 0$, $x \le 0$, $-4\dot{x} + 4\dot{y} \le 0$, $-\dot{x} - 2\dot{y} \le 0$, $l_6$
$x - y = 0$, $y \le 0$
$y \le 0$, $x - y \le 0$, $-5\dot{x} + 2\dot{y} \le 0$, $4\dot{x} - 4\dot{y} \le 0$, $l_5$

(c)

**Figure 5.1: Example 5.2. (a) Conic partition of state space. (b) Conic differential inclusion. (c) Conic abstraction of the system.**

differential inclusion of each location are polyhedral cones. □ Similarly as for the symbolic reachability analysis of LHA [6], the set of states that are reachable from an initial set $X \subseteq \mathbb{R}^n$ through the continuous flow at location $l_C \in L$ corresponding to $C \in \Delta$ is given by

$$(X \oplus \boldsymbol{A}C) \cap C. \tag{5.1}$$

A conic abstraction represents an overapproximation of the system, whose tightness depends on the maximum angle between any two points in the cone $\boldsymbol{A}C$ in derivative space. Roughly speaking, the more acute the cone $\boldsymbol{A}C$ in derivative space, the more accurate the overapproximation. Figure 5.2a shows a comparison between conic partitions with different accuracies (depicted in two different shades) for the same initial region. We encapsulate the accuracy given by a partition with the notion of twisting.

**Definition 13** (Twisting of a state region)**.** Let $\dot{\vec{x}} = \boldsymbol{A}\vec{x}$ be a linear system and $P \subseteq \mathbb{R}^n$ be a (not necessarily conic) region of the state space. Then $P$ is said to have a twisting of $\theta$ (or to be $\theta$ twisted) if it satisfies that

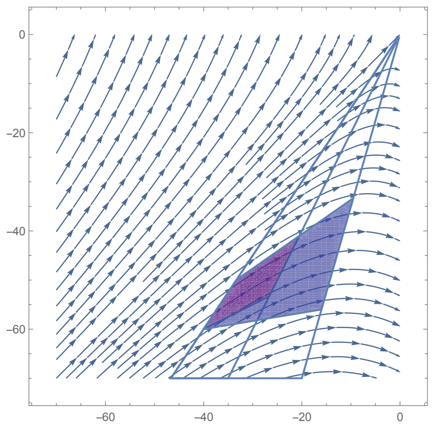
$$\sup_{\vec{x}_1, \vec{x}_2 \in P} \arccos\left(\frac{\langle \dot{\vec{x}}_1, \dot{\vec{x}}_2 \rangle}{\|\dot{\vec{x}}_1\| \|\dot{\vec{x}}_2\|}\right) = \theta. \tag{5.2}$$

Intuitively, a cone with smaller twisting allows only trajectory segments that are almost straight, inducing a more accurate overapproximation. In the context of conic abstraction, properly inducing smaller and smaller twistings induce refinements of the abstraction, providing a better overapproximation.

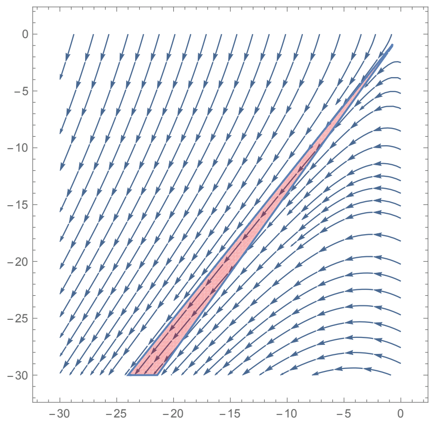
**Definition 14** (Conic abstraction refinement)**.** Given two conic abstractions $(L_1, E_1)$ and $(L_2, E_2)$ for a linear system $\dot{\vec{x}} = \boldsymbol{A}\vec{x}$, $(L_2, E_2)$ refines $(L_1, E_1)$ if $|L_2| > |L_1|$ and for all $l_1 \in L_1$ with cone $C_1$ there does always exist $l_2^1, \ldots, l_2^m \in L_2$ with cones $C_2^1, \ldots, C_2^m$ such that $C_1 = C_2^1 \cup \cdots \cup C_2^m$.

It is subject of Sec. 5.3 how to generate abstraction refinements by tuning the value of twisting.

The property we desire is that the twisting of every state partition is bounded by a small angle $\theta$. A common strategy to achieve this goal is to split the state space into small rectangles iteratively until the twisting of each rectangle falls below $\theta$ [68; 49; 52]. However, such strategy is inefficient, as the twisting may not change uniformly in a rectangular partition. On the contrary, a conic partition naturally enjoys bounded twisting using unbounded regions. This allows a conic partition to accurately overapproximate both bounded and unbounded reach pipes, if in the latter case the trajectories are straight enough. Figure 5.2b shows such an example, where the tiny cone overapproximates all trajectories entering it, as they tend to be parallel to its left boundary.



**(a)**



**(b)**

**Figure 5.2: (a) Overapproximation inside different cones. The smaller the cone, the more precise the overapproximation. (b) A cone capable of offering accurate overapproximation for unbounded reach pipe.**

### 5.2.1 Conic abstractions derived from derivative space partitions

In existing work on discrete abstraction of continuous systems, to obtain a high-quality state space partition, the focus is mostly placed on state space. However, what really matters here is the derivative space. Therefore, our state space partition should be derived from a derivative space partition. Given a continuous system $\dot{\vec{x}} = f(\vec{x})$, every convex cone $D$ in the derivative space with a maximal angle $\theta$ corresponds to a set $C$ of states which has a twisting of $\theta$. Moreover, $C$ can be obtained through simple substitution. However, for nonlinear systems, $C$ is nonlinear and is hard to handle, so we leave it for future work.

We assume that the systems under consideration are linear. To derive a conic abstraction for an $n$-dimensional linear system, we first partition the whole derivative space into a set $\Omega$ of convex polyhedral cones which satisfies that

1. $\bigcup_{D_i \in \Omega} D_i = \mathbb{R}^n$;
2. $\forall D_i, D_j \in \Omega : (D_i \backslash \partial D_i) \bigcap (D_i \backslash \partial D_j) = \emptyset$;
3. $\forall D_i \in \Omega : \sphericalangle D_i \leq \theta$, where $\sphericalangle D_i$ denotes the maximal angle of $D_i$ (i.e. the maximal angle between the vectors in $D_i$) and $\theta$ is a given bound.

By mapping $\Omega$ back to the state space, we can obtain another set $\Delta$ of state regions. The property of $\Delta$ is formalized in the following theorem.

**Theorem 5.2.1.** *Given a linear system $\dot{\vec{x}} = \boldsymbol{A}\vec{x}$ let $\Omega$ be a set of convex polyhedral cones defined as above and $\Delta = \{\boldsymbol{A}^{-1}D \mid D \in \Omega\}$. Then,*

1. *every $C_i \in \Delta$ is a convex polyhedral cone and the twisting of $C_i$ is $\theta$-bounded;*
2. *$\bigcup_{C_i \in \Delta} C_i = \mathbb{R}^n$;*
3. *$\forall C_i, C_j \in \Delta : (C_i \backslash \partial C_i) \bigcap (C_i \backslash \partial C_j) = \emptyset$;*

*Proof.* 1. Let $C_i = \boldsymbol{A}^{-1}D_i \in \Delta$, we first prove $C_i$ is a convex polyhedral cone. Given any two vectors $\vec{x}_1, \vec{x}_2 \in C_i$, we only need to prove that $\alpha\vec{x}_1 + \beta\vec{x}_2 \in C_i$ for any $\alpha > 0, \beta > 0$. Since $\vec{x}_1, \vec{x}_2 \in C_i$, we have $\boldsymbol{A}\vec{x}_1 \in D_i, \boldsymbol{A}\vec{x}_2 \in D_i$, then $\alpha\boldsymbol{A}\vec{x}_1 + \beta\boldsymbol{A}\vec{x}_2 = \boldsymbol{A}(\alpha\vec{x}_1 + \beta\vec{x}_2) \in D_i$ because $D_i$ is a convex polyhedral cone. Therefore, $\alpha\vec{x}_1 + \beta\vec{x}_2 \in \boldsymbol{A}^{-1}D_i = C_i$. Moreover, since $\boldsymbol{A}C_i = D_i$ and $\sphericalangle D_i \leq \theta$, the twisting of $C_i$ must be $\theta$-bounded.

2. Since $\boldsymbol{A}$ is full rank, then the mapping $\boldsymbol{A}^{-1}\vec{x}$ is a bijection. Hence, $\mathbb{R}^n = \boldsymbol{A}^{-1}\mathbb{R}^n = \boldsymbol{A}^{-1}(\bigcup_{D_i \in \Omega} D_i) = \bigcup_{C_i \in \Delta} C_i$.

3. Since $\forall D_i, D_j \in \Omega : (D_i \backslash \partial D_i) \bigcap (D_i \backslash \partial D_j) = \emptyset$, by applying $\boldsymbol{A}^{-1}$ to the above formula, we can derive that $\forall C_i, C_j \in \Delta : (C_i \backslash \partial C_i) \bigcap (C_i \backslash \partial C_j) = \emptyset$ holds.

□

According to Theorem 5.2.1, we know that, given any linear system $\mathcal{H}$ with an invertible matrix $\boldsymbol{A}$ and a $\theta$-bounded conic partition $\Omega$ of the derivative space, a conic partition $\Delta$ for the state space with $\theta$-bounded twisting can be obtained by a linear transformation. Note that the twisting of $C_i$ is $\theta$-bounded does not mean that $C_i$ is $\theta$-bounded. Conversely, the maximal angle of each cone $C_i$ varies significantly depending on how straight the trajectories are in that cone. Roughly speaking, the straighter the trajectories are, the larger the maximal angle of $C_i$ is, provided that the twisting is the same. □

Now, let us get back to the issue of generating a conic partition of the derivative space. Our approach borrows the idea of slicing watermelons. Concretely, given an $n$-dimensional derivative space, we first choose a group of seed planes passing through the origin and then generate a cluster of planes by rotating each seed plane counterclockwise around an

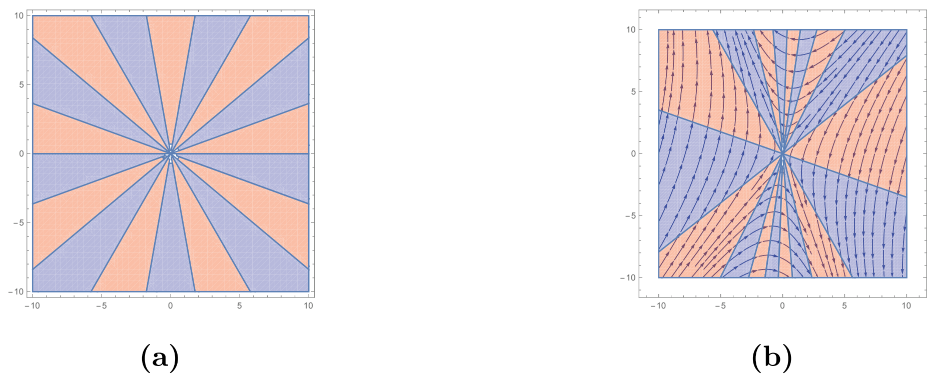


**Figure 5.3: Example 5.2.1 (a) Uniformly conic partition of the the derivative space. (b) Conic partition of state space derived from the derivative space partition.**

independent axis by a fixed angle $\theta_1$, step by step until no further $\theta_1$ rotation is possible. Finally, the whole vector space can be sliced into a set of convex polyhedral cones by the generated planes and each of them is $\theta_2$-bounded for some $\theta_2$. By mapping these cones into the state space, we can achieve a conic partition of $\theta_2$-bounded twisting for the state space. The following example shows how a conic state space partition derived from a uniform derivative space partition looks like.

Consider the following linear system $\mathcal{H}$ described by $\dot{x} = -2x - 2y, \dot{y} = -5x + y$. As shown in Figure 5.3a, the derivative space is first uniformly partitioned into 18 cones. Then, these cones are mapped into the state space. As can be seen in Figure 5.3b, in every cone, the straighter the trajectories are, the larger the maximal angle of the cone is. $\square$

The reachable set computation of a conic abstraction is a basic operation of linear hybrid automata. As usual, due to the undecidable nature of the issue, the reachable set computation of a conic abstraction cannot guarantee to terminate for a general linear system. However, for the conic abstraction of a specific class of systems, the reachable set computation can be guaranteed to terminate, which is shown in the next section.

## 5.3 Diagonalizable Systems

In this section, we focus on a class of affine systems for which the matrix used to describe the system dynamics is diagonalizable in $\mathbb{R}$, called diagonalizable systems. The reason why diagonalizable systems are interesting is that, given a conic abstraction, the reachable set computation is guaranteed to terminate. Formally, a diagonalizable system is defined as follows.

**Definition 15** (Diagonalizable system)**.** An affine system $\dot{\vec{x}} = \boldsymbol{A}\vec{x} + \vec{b}$ is diagonalizable if there exist a real matrix $Q$ such that $Q^{-1}\boldsymbol{A}Q = \mathbf{diag}(\lambda_1, \cdots, \lambda_n)$, where $\lambda_i \in \mathbb{R}, \lambda_i \neq 0, i = 1, \cdots, n$.

In the following, we introduce how to derive a conic abstraction for a diagonalizable system and how to overapproximate their reachable sets by the conic abstraction. We also extend the theory to hybrid affine systems.

### 5.3.1 Properties of diagonalizable systems

The most important feature of diagonalizable system is that all of their eigenvalues are real numbers. Given a diagonalizable affine system $\dot{\vec{x}} = \boldsymbol{A}\vec{x} + \vec{b}$ with initial region $X_0$, by doing a translation on the coordinate system with $\vec{y} = \vec{x} + \boldsymbol{A}^{-1}\vec{b}$, we can always transform the system into a linear system $\dot{\vec{y}} = \boldsymbol{A}\vec{y}$ with initial region $Y_0 = X_0 \oplus \{\boldsymbol{A}^{-1}\vec{b}\}$. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of $\boldsymbol{A}$ and $\vec{u}_1, \ldots, \vec{u}_j$ be the corresponding eigenvectors respectively, then the general solution of the linear system can be written as (refer to [72])

$$\vec{x}(t) = c_1 e^{\lambda_1 t}\vec{u}_1 + \cdots + c_n e^{\lambda_n t}\vec{u}_n \tag{5.3}$$

where $c_1, \ldots, c_n$ depends on the initial value $\vec{x}_0$ of the system of differential equations and can be obtained by solving $\vec{x}(0) = \vec{x}_0$. Let $\boldsymbol{U} = (\vec{u}_1, \ldots, \vec{u}_n)$ and $\vec{c} = (c_1, \ldots, c_n)$, $Cone(\vec{c}, \boldsymbol{U}) = \{\vec{x} \in \mathbb{R}^n \mid \vec{x} = \sum_{i=1}^{n} t_i c_i \vec{u}_i, t_i \geq 0\}$ denote the convex polyhedral cone generated by the vectors $c_1\vec{u}_1, \ldots, c_n\vec{u}_n$. Then, we have the following theorem.

**Theorem 5.3.1.** *Given a diagonalizable system $\dot{\vec{x}} = \boldsymbol{A}\vec{x} + \vec{b}$, let $\boldsymbol{U}$ be defined as above and $\Xi = \{-1, 1\}^n$. Then, for every $\vec{\xi} \in \Xi$, $Cone(\vec{\xi}, \boldsymbol{U})$ is an invariant and the twisting of $Cone(\vec{\xi}, \boldsymbol{U})$ is bounded by radian $\pi$.*

*Proof.* For system $\mathcal{H}_2$, given any initial point $\vec{x}_0$, there must exist a set of constants $\vec{c} = (c_1, \ldots, c_n)$ such that the solution for the initial value $\vec{x}_0$ satisfies the equation (5.3) and $\vec{x}(0) = \sum_{i=1}^{n} c_i \vec{u}_i$. Let $\vec{\xi} = (sign(c_1), \ldots, sign(c_n))$, then $\vec{x}(0) \in Cone(\vec{\xi}, \mathrm{U})$, where $sign(c_i) = 1$ if $c_i \geq 0$ otherwise $sign(c_i) = -1$. Note that $\vec{\xi}$ could be any value in $\mathit{\Xi}$ depending on $\vec{x}_0$. Since for any $t \geq 0$, we have $e^{\lambda_i t} > 0$, which means that $sign(c_i) = sign(c_i e^{\lambda_i t})$. Therefore, $\forall t \geq 0 : \vec{x}(t) \in Cone(\vec{\xi}, \mathrm{U})$, i.e. $Cone(\vec{\xi}, \mathrm{U})$ is an invariant of the system. Since $Cone(\vec{\xi}, \mathrm{U})$ is a convex polyhedral cone, $\boldsymbol{A}Cone(\vec{\xi}, \mathrm{U})$ is a convex polyhedral cone as well, hence the maximal angle of $\boldsymbol{A}Cone(\vec{\xi}, \mathrm{U})$ must be bounded by $\pi$. Correspondingly, the twisting of $Cone(\vec{\xi}, \mathrm{U})$ is bounded by $\pi$. $\square$

According to Theorem 5.3.1, the state space of a diagonalizable system can always be partitioned into a set of invariant cones and the twisting of every invariant cone is bounded by radian $\pi$. Therefore, given a diagonalizable system, to overapproximate the reachable set, we do not have to construct a conic abstraction for the whole state space. Instead, we only need to figure out which invariant cones the initial set spans and then construct a conic abstraction for each of them respectively. As mentioned previously, we would start from partitioning the derivative space. Based on the property of diagonalizable system, we develop a partitioning scheme which can construct a conic abstraction as a directed acyclic graph.

### 5.3.2 Diagonalization and conic partition

The first step of constructing a conic partition consists of diagonalizing the original system. Given a diagonalizable system $\dot{\vec{y}} = \boldsymbol{A}\vec{y}$ with initial region $Y_0$, a diagonalization of it is a linear system $\dot{\vec{z}} = \boldsymbol{A}_\lambda \vec{z}$ with initial region $Z_0$ where $\boldsymbol{A}_\lambda = \boldsymbol{Q}^{-1}\boldsymbol{A}\boldsymbol{Q}$ is a diagonal matrix and $Z_0 = \boldsymbol{Q}^{-1}Y_0$ for some $\boldsymbol{Q}$. In theory, the diagonalized system is equivalent to the original system in terms of safety verification. However, by doing diagonalization, we manage to transform every invariant cone and its derivative cone into an independent orthant respectively. Since an orthant as a cone has some good properties such as having a fixed maximal angle of $\frac{\pi}{2}$ and all the generating vectors of the invariant cones are orthogonal to each other, we propose a special conic partition scheme, called *radial partition*, which can result in a directed acyclic graph for the conic abstraction.

Given a diagonalized $n$-dimensional system $\dot{\vec{z}} = \boldsymbol{A_\lambda}\vec{z}$ and an orthant $\boldsymbol{O} = \{\vec{z} \in \mathbb{R}^n \mid \boldsymbol{B}\vec{z} \le \vec{0}\}$ in derivative space, where $\boldsymbol{B} = diag(b_{11}, \ldots, b_{nn})$ with $b_{ii} = 1$ or $-1$. Let $\boldsymbol{B}_i, \boldsymbol{B}_j$ be the $i$'th and $j$'th row vectors of $\boldsymbol{B}$ respectively, where $i \neq j$. The basic idea of *radial partition* is as follows. For every pair of $(\boldsymbol{B}_i, \boldsymbol{B}_j)$, we generate a sequence of vectors $S_{ij} : \vec{v}_{ij1}, \ldots, \vec{v}_{ij(K_{ij}+1)}$ by rotating the vector $\vec{v}_{ij1} = \boldsymbol{B}_j$ from $\boldsymbol{B}_i$ to $\boldsymbol{B}_j$ step by step with an rotating amplitude $\frac{\pi}{2K_{ij}}$. Then, $S_{ij}$ is used as the sequence of normal vectors of partitioning planes. Thus, each pair of adjacent vectors $\vec{v}_{ijk}, \vec{v}_{ijk+1}$ forms a slice $\{\vec{z} \in \mathbb{R}^n \mid \langle \vec{v}_{ijk}, \vec{z}^T \rangle \le 0, \langle -\vec{v}_{ijk+1}, \vec{z}^T \rangle \le 0\}$ of the orthant $O$ and $O$ will be partitioned into $K_{ij}$ slices by all the planes formed by $S_{ij}$. Hence, we can get $\frac{n(n-1)}{2}$ ordered sequences of planes at most totally. These planes intersecting each other yield a conic partition $D$ for the orthant $O$. However, we do not really need so many sequences of partitioning planes. Actually, $n-1$ sequences of planes suffices to construct a partition with an arbitrarily small maximal angle.

For the conic abstraction derived from radial partition, we have the following theorem.

**Theorem 5.3.2.** *Every conic abstraction derived from a radial partition of the derivative space is a directed acyclic graph.*

*Proof.* Let $S_{ij} : \vec{v}_{ij1}, \ldots, \vec{v}_{ij(K_{ij}+1)}$ and $O$ be defined as in Subsection 5.3.2, $P_{ij} : \vec{w}_{ij1}, \ldots, \vec{w}_{ij(K_{ij}+1)}$ be the ordered sequence obtained from $S_{ij}$ with $\vec{w}_{ijk} = \vec{v}_{ijk}\boldsymbol{A}_\lambda$ and $O_1 = \{\vec{x} \in \mathbb{R}^n \mid \boldsymbol{B}\boldsymbol{A}_\lambda \le 0\}$ be the orthant in state space corresponding to $O$, which means that all the $P_{ij}$ define the conic partition for $O_1$. Then, we only need to prove that, for every sequence $P_{ij}$, all the trajectories point in the same side on all but the first and last partitioning planes (note that $\vec{w}_{ij1}, \vec{w}_{ij(K_{ij}+1)}$ denote the boundaries of $O_1$), or equivalently,

$$\begin{aligned} \forall k \in [2, K_{ij}] : \forall \vec{x} \in O_1 :& (\langle \vec{w}_{ijk}, \vec{x} \rangle = 0 \implies \mathcal{L}_{\boldsymbol{A}_\lambda} \langle \vec{w}_{ijk}, \vec{x} \rangle \sim 0) \implies \\ & (\langle \vec{w}_{ij(k+1)}, \vec{x} \rangle = 0 \implies \mathcal{L}_{\boldsymbol{A}_\lambda} \langle \vec{w}_{ij(k+1)}, \vec{x} \rangle \sim 0) \end{aligned} \tag{5.4}$$

where $\sim \in \{>,<\}$. Let $\vec{v}_{ij1} = B_j$ and the rotation matrix associated with $B_i, B_j$ be (see [62])

$$R_{ij}(\theta) = \left[\begin{array}{c|cccccccc} & & & i & & j & & \\ & 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ & \vdots & \ddots & \vdots & \cdots & \vdots & \cdots & \vdots \\ i & 0 & \cdots & \cos(\theta) & \cdots & -\sin(\theta)b_{ii}b_{jj} & \cdots & 0 \\ & \vdots & \cdots & \vdots & \ddots & \vdots & \cdots & \vdots \\ j & 0 & \cdots & \sin(\theta)b_{ii}b_{jj} & \cdots & \cos(\theta) & \cdots & 0 \\ & \vdots & \cdots & \vdots & \cdots & \vdots & \ddots & \vdots \\ & 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{array}\right] \tag{5.5}$$

Then, we can generate all $\vec{v}_{ijk}$ iteratively using the formula $\vec{v}_{ijk+1} = \vec{v}_{ijk}R_{ij}^{-1}$, where $R_{ij}^{-1}(\theta) = R_{ij}(-\theta)$ and the general representation of $\vec{v}_{ijk}$ is as follows.

$$\vec{v}_{ijk} = (0,\ldots,-b_{ii}\sin((k-1)\theta),\ldots,b_{jj}\cos((k-1)\theta)),\ldots,0) \tag{5.6}$$

Let $\boldsymbol{A}_{\vec{\lambda}} = diag(\lambda_1,\ldots,\lambda_n)$, then the general representation of $\vec{w}_{ijk} = \vec{v}_{ijk}A_\lambda$ and $\mathcal{L}_{A_\lambda}\langle\vec{w}_{ijk},\vec{x}\rangle = \vec{w}_{ijk}A_\lambda$ are as follows respectively.

$$\vec{w}_{ijk} = (0,\ldots,-\lambda_i b_{ii}\sin((k-1)\theta),\ldots,\lambda_j b_{jj}\cos((k-1)\theta)),\ldots,0) \tag{5.7}$$

$$\mathcal{L}_{A_\lambda}\langle\vec{w}_{ijk},\vec{x}\rangle = (0,\ldots,-\lambda_i^2 b_{ii}\sin((k-1)\theta),\ldots,\lambda_j^2 b_{jj}\cos((k-1)\theta)),\ldots,0) \tag{5.8}$$

Assume $\forall\vec{x}\in O_1 : \langle\vec{w}_{ijk},\vec{x}\rangle = 0 \implies \mathcal{L}_{\boldsymbol{A}_\lambda}\langle\vec{w}_{ijk},\vec{x}\rangle > 0$ holds, by simplification, we get $\forall\vec{x}\in O_1 : \lambda_j(\lambda_j-\lambda_i)b_{jj}x_j\cos((k-1)\theta) > 0$. Now, assume $\langle\vec{w}_{ijk+1},\vec{x}\rangle = 0$, we can derive $\mathcal{L}_{\boldsymbol{A}_\lambda}\langle\vec{w}_{ijk+1},\vec{x}\rangle = \lambda_j(\lambda_j-\lambda_i)b_{jj}x_j\cos(k\theta)$. Since $0 < k\theta < \frac{\pi}{2}$, then $\cos((k+1)\theta) > 0$ and $\cos((k+1)k\theta) > 0$. Hence, $\forall\vec{x}\in O_1 : \mathcal{L}_{\boldsymbol{A}_\lambda}\langle\vec{w}_{ijk+1},\vec{x}\rangle > 0$. Similarly, the case of $\mathcal{L}_{\boldsymbol{A}_\lambda}\langle\vec{w}_{ijk},\vec{x}\rangle < 0$ can also be proved. Therefore, the theorem holds. □

By Theorem 5.3.2, the reachable set exploration of the conic abstraction derived from a radial partition is guaranteed to terminate. Moreover, as indicated in the proof, the direction of the discrete transition between locations can be easily determined by the sign of the Lie derivatives of the partitioning planes at the beginning [79]. □

## 5.4 Time-unbounded Reachability Analysis

In this section, we present how to compute the overapproximation of reach pipe of a given affine system based on the conic abstraction.

**Algorithm 1:** Reach pipe overapproximation of affine systems

```
input  : System $\dot{\vec{x}} = \boldsymbol{A}\vec{x} + \vec{b}$ and initial set $X_0$;
local  : Heap of partition regions $H$; /*stores unique elements only*/
output : Map from partition region to polyhedron $R$; /*by default maps to $\emptyset$*/

1  $\boldsymbol{A}_{\vec{\lambda}} \leftarrow \boldsymbol{Q}^{-1}\boldsymbol{A}\boldsymbol{Q}$; /*diagonalize*/
2  $Z_0 \leftarrow \boldsymbol{Q}^{-1}(X_0 \oplus \{\boldsymbol{A}^{-1}\vec{b}\})$; /*transform into linear system and diagonalize*/
3  foreach $C$ partition region in state space such that $Z_0 \cap C \neq \emptyset$ do
4      insert into R($C$) the template polyhedron of $[(Z_0 \cap C) \oplus \boldsymbol{A}_{\vec{\lambda}}C] \cap C$;
5      push $C$ into H;
6  end
7  while $H$ is not empty do
8      $C \leftarrow$ pop the top of H;
9      foreach $D$ successor partition region of $C$ such that $R(C) \cap D \neq \emptyset$ do
10         join R($D$) with the template polyhedron of $[(R(C) \cap D) \oplus \boldsymbol{A}_{\vec{\lambda}}D] \cap D$;
11         push $D$ into H;
12     end
13 end
```

We first diagonalize the system (as in Sec. 5.3.2) and we identify the regions that hit the initial region. Then we iteratively explore the adjacent regions, while computing and storing the reach pipe. In particular, we build the control graph of the conic abstraction incrementally and only for those locations that are indeed reachable. We outline our procedure in Algorithm 1.

- The first two lines aim to translate the equilibrium point to the origin and further diagonalize the system. The initial set $X_0$ undergoes a similar transformation.

- In line 3–6, we split the initial set into multiple regions. For each split, we compute the overapproximation of the reach pipe inside the respective region, as defined in Eq. 5.1. We store the result in $R$ and we push the region to $H$ for further exploration.

- In line 7–13, we compute the overapproximation of following reach pipes inside the adjacent regions. The result is joined to what previously computed in the same region. The join consists of taking a convex hull between template polyhedra. Each such successor region is pushed to $H$.

We optimize the exploration order so to explore the successors of a specific region at most once, namely we want the heap $H$ to never pop a region twice at line 8. To this aim, we instruct $H$ to maintain a topological order between regions given by the graph of the conic abstraction (see Def. 12). Such order always exists, as a radial partition always induces an acyclic one (see Thm. 5.3.2). Similarly, on the enumeration of line 9, each region $D$ must satisfy the same order w.r.t. $C$. Concretely, the order between regions is the closure of the order given by the Lie derivative of their common facets (as in Def. 12).

We produce a map from partition regions to template polyhedra, where each template polyhedron overapproximates the reach pipe at the respective region. Precisely, the template polyhedron of a convex set $X \subseteq \mathbb{R}^n$ w.r.t. the finite set of directions $D \subseteq \mathbb{R}^n$, which we call the template, is the tightest polyhedron enclosing $X$ whose facets are normal to all and only the directions in $D$. We efficiently compute the template polyhedra at lines 4 and 10 using linear programming [104] and the convex hull at line 10 by simply taking for each direction the facet that is the loosest between the two. The choice of template is critical for the quality of the abstraction and the efficiency of the procedure. In each region we use the octagonal template, augmented with the normals of the facets of both the derivative and the state space cones.

In the following, we exemplify the result of the procedure on our running example under different granularities of the partition. Consider the system in Example 5.2, let the initial set be $X_0 = \{(x,y) \in \mathbb{R}^2 \mid -30 \leq x \leq -28, -45 \leq y \leq -43\}$ and the invariant be $\mathbb{R}^2$. We diagonalize and transform the system dynamics into $\dot{x} = -4x, \dot{y} = 3y$ with initial state $Z_0 = \{(x,y) \in \mathbb{R}^2 \mid -x + \frac{2}{5}y \leq 30, x - \frac{2}{5}y \leq -28, -x - y \leq 45, x + y \leq -43\}$. By partitioning the orthant into 5, 20 and 60 cones respectively, we got 3 overapproximations of different accuracies for the unbounded reachable set, which is shown in Figure5.4. As can be seen, the precision of the overapproximation increases rapidly with the number of cones. □

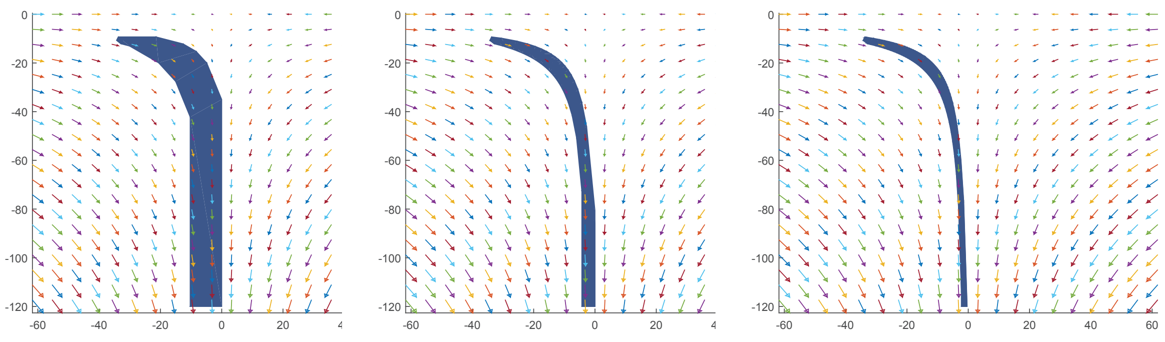

(a) $K = 5$ (b) $K = 20$ (c) $K = 60$

**Figure 5.4: Unbounded invariants obtained for Example 13 under different numbers of slices of partition.**

### 5.4.1 Mode switching

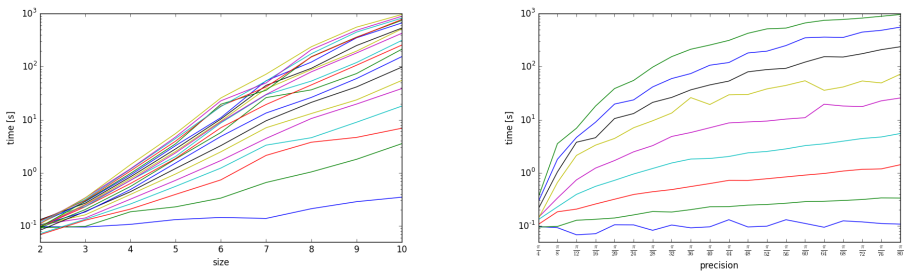
The theory presented in the previous sections can be easily extended to deal with hybrid systems. Given a hybrid system, the conic abstraction of each discrete location can be done as presented. However, due to the transformation of the system dynamics in each location, the same transformation also needs to be applied to the guards and reset operations of the discrete transitions between locations.

Concretely, let $\dot{\vec{y}} = \boldsymbol{A}_i\vec{y} + \vec{b}_i$ and $\dot{\vec{y}} = \boldsymbol{A}_j\vec{y} + \vec{b}_j$ be the dynamics of two discrete locations $l_i, l_j$ of a hybrid system, $G_{ij} = \{\vec{y} \in \mathbb{R}^n \mid \boldsymbol{J}_{ij}\vec{y} \leq \vec{h}_{ij}\}$ be the guard of the transition $(l_i, l_j)$ and $T_{ij} : \vec{y}' \mapsto \boldsymbol{M}_{ij}\vec{y} + \vec{e}_{ij}$ be the reset operation. Suppose the diagonalization of $\boldsymbol{A}_i, \boldsymbol{A}_j$ are $\boldsymbol{A}_{\vec{\lambda}_i} = \boldsymbol{Q}_i^{-1}\boldsymbol{A}_i\boldsymbol{Q}_i, \boldsymbol{A}_{\vec{\lambda}_j} = \boldsymbol{Q}_j^{-1}\boldsymbol{A}_j\boldsymbol{Q}_j$, respectively. Let $\vec{x}$ be the variable name after transformation, then we have $l_i : \vec{y} = \boldsymbol{Q}_i\vec{x} + \boldsymbol{A}_i^{-1}\vec{b}_i$ and $l_j : \vec{y} = \boldsymbol{Q}_j\vec{x} + \boldsymbol{A}_j^{-1}\vec{b}_j$. Thus, the guard and the reset operation are transformed into the following.

$$G^*_{ij} = \{\vec{x} \in \mathbb{R}^n \mid \boldsymbol{J}_{ij}\boldsymbol{Q}_i\vec{x} \leq \vec{h}_{ij} - \boldsymbol{J}_{ij}\boldsymbol{A}_i^{-1}\vec{b}_i\} \tag{5.9}$$

$$T^*_{ij} : \vec{x} \mapsto \boldsymbol{Q}_j^{-1}(\boldsymbol{M}_{ij}\boldsymbol{Q}_i\vec{x} + \boldsymbol{M}_{ij}\boldsymbol{A}_i^{-1}\vec{b}_i + \vec{e}_{ij} - \boldsymbol{A}_j^{-1}\vec{b}_j) \tag{5.10}$$

Location invariants $I_j$ are transformed as well using the formula $I'_j = \{\vec{x} \in \mathbb{R}^n \mid \vec{x} = \boldsymbol{Q}^{-1}(\vec{y} + \boldsymbol{A}^{-1}\vec{b}), \vec{y} \in I_j\}$. By applying the above transformations to the whole hybrid system and then performing the conic abstraction, we obtain an LHA, whose reachability analysis can be done as usual. However, unlike for pure continuous systems, termination is not guaranteed.



(a) (b)

**Figure 5.5: Scalability of our method in computing the abstraction of purely continuous systems. The abscissa of (a) refer to the number of variables and each curve refers to a precision (maximum angle), while the abscissa of (b) refers to precisions and each curve refers to a system size (# variables). Both ordinates show the average runtime for 50 randomly generated systems for each system size and precision.**

## 5.5 Experiments

We have implemented the procedure presented above in C++ using the GLPK library for linear programming [1], and we have performed two experiments. In the first, we have performed a scalability test using purely continuous systems given by random matrices of increasing size and for increasing precision of the analysis. In the second, we have considered the room heating benchmark and compared against *SpaceEx* under scenarios `supp` and `stc` and *PHAVer* [86; 56; 52].

We generated random diagonal matrices with non-zero distinct integer values between -10 and 10 on the diagonal. Then we measured the runtime of our procedure for the maximum angles of $\frac{\pi}{2k}$ for increasing $k$ (two by two) and the initial state being a unit box centered in $(50, \ldots, 50)$. Figure 5.5a shows that the runtime increases exponentially with the number of variables, while the more the precision increases the less (for fixed system size) the difference in runtime is affected. The latter is also confirmed by Fig. 5.5b, which shows that the runtime increases polynomially with the increase in precision and that the number of variables affects the degree of the polynomial as, in fact, the number of partitions is worst case $k^n$.

| | Time part. | | Conic part. | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *SpaceEx* | | *PHAVer* | | | | Our method | | | |
| | `supp` | `stc` | $\pi/4$ | $\pi/20$ | $\pi/40$ | $\pi/80$ | $\pi/4$ | $\pi/20$ | $\pi/40$ | $\pi/80$ |
| `heat-2` | err | oot | 0.17 | 2.20 | 9.86 | 50.86 | 0.24 | 0.25 | 0.31 | 0.41 |
| `heat-3` | err | oot | oot | oot | oot | - | 147 | 2.41 | 5.18 | 12.32 |
| `heat-4` | err | oot | oot | - | - | - | 14155 | 278 | 190 | 1217 |
| `heat-5` | err | oot | oot | - | - | - | oot | oot | 27467 | 56671 |
| `heat-6` | err | oot | - | - | - | - | oot | oot | oot | oot |

**Table 5.1: Runtimes for the abstraction of the room heating benchmark with 2 to 6 rooms. *SpaceEx* has been run with scenarios `supp` and `stc`, template `oct`, and time horizon of 1. *PHAVer* has been run on explicit conic partitions for the given precisions whose generation time is excluded here. We used a 2.6 GHz CPU with 4Gb RAM. The key err indicates error, oot out of time (24 hours), and - experiment not executed, i.e., the explicit partitioning run out of 24 hours time.**

The room heating benchmark describes a protocol for heating a number of rooms with a limited number of shared heaters [51]. We consider houses with 2 to 6 ordered rooms, each room is only adjacent to the previous and the following room, and all but one room have a heater. The temperature of room $i$ is governed by a linear ODE of the form

$$\dot{x}_i = ch + b_i(u - x_i) + \sum_{j \neq i} a_{ij}(x_j - x_i) \tag{5.11}$$

where $c$ is the heater efficiency, $h$ indicates whether the heater is present, $b_i$ is the room dispersion, $u$ is external temperature, and $a_{ij}$ is the heat exchange between rooms ($a_{ij} = 0$ for non-adjacent rooms). The switching logic moves a heater from a room to an adjacent room if the temperature difference exceeds a threshold and the latter is colder. In addition, we augmented every mode with a dummy self switch, so to force *SpaceEx* to perform time-unbounded reachability.

We have verified the room heating benchmark using *SpaceEx* with both scenarios `supp` and `stc` and in both cases it either crashes or timeouts. Conversely, using *PHAVer* the procedure terminated, but for small models only. Similarly to our method, *PHAVer* abstracts affine systems into LHA, but it requires the user to provide an explicit partition

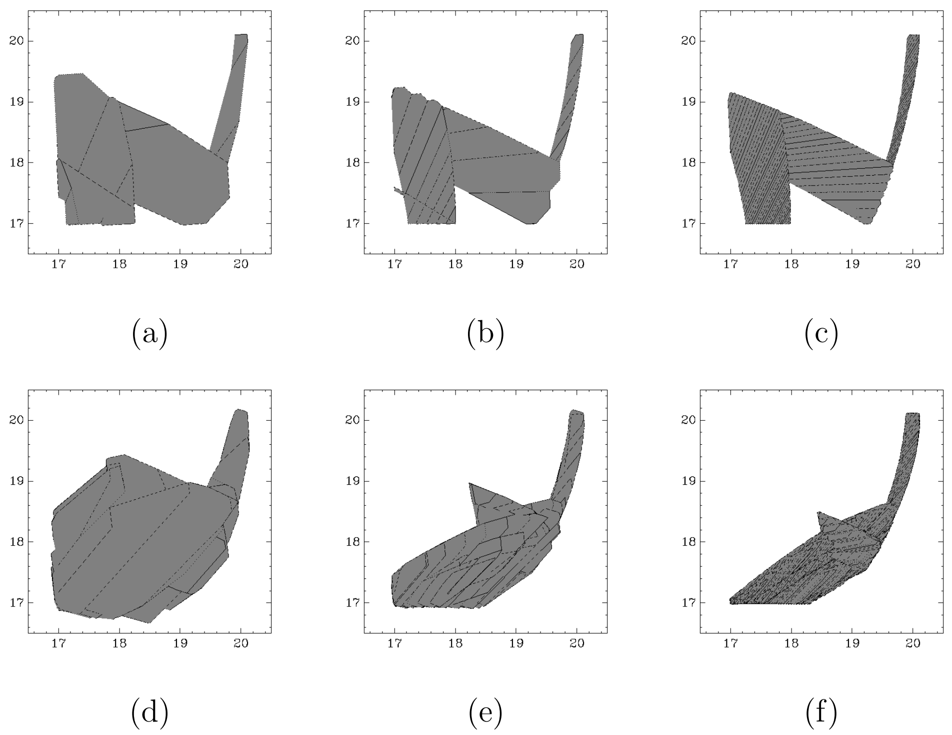


**Figure 5.6: Conic abstractions of the heating benchmark for 2 rooms (a, b, and c) and 2-dimensional projection for 3 rooms (d, e, and f) for resp. precisions $\pi/20$ (a and d), $\pi/80$ (b and e), and $\pi/400$ (c and f).**

of the state space (rather than the derivative space). We have generated equivalent conic abstractions in the form of explicit LHA and verified them with *PHAVer*. Note that if such LHA is not provided, *PHAVer* computes trivial abstractions by using the whole mode invariants as partitions. *PHAVer* uses quantifier elimination for forward reachability, while we compute template polyhedra.

The time results are shown in Tab. 5.1. First, PHAver always times out for systems with more than 2 variables and even for 2-dimensional it scales poorly in precision compared to our method. Second, beyond three dimensional systems our method is even faster than generating the explicit LHA. The scalability in dimensionality indicated the advantage of using template polyhedra rather quantifier elimination while the scalability in precision demonstrates the advantage of using our incremental construction of the conic partition. Figure 5.6 depicts the abstractions for the 2 and 3 rooms systems and for precisions $\pi/20$, $\pi/80$, and additionally $\pi/400$, computed using our method. Predictably, one can see how the quality of the abstraction increases as the precision increases.

# 6 Space-Time Interpolation for Affine Hybrid Automata

Formal verification techniques can be used to either provide rigorous guarantees about the behaviors of a critical system, or detect instances of violating behavior if such behaviors are possible. Formal verification has become widely used in the design of software and digital hardware, but has yet to show a similar success for physical and cyber-physical systems. One of the reasons for this is a scarcity of suitable algorithmic verification tools, such as model checkers, which are formally sound, precise, and scale reasonably well. In this paper, we propose a novel verification algorithm that meets these criteria for systems with piecewise affine dynamics. The performance of the approach is illustrated experimentally on a number of benchmarks. Since systems with affine dynamics have been studied before, we first describe why the available methods and tools do not handle this class of systems sufficiently well, and then describe our approach and its core contributions.

**Previous approaches** The algorithmic verification of systems with continuous or discrete-continuous (hybrid) dynamics is a hard problem both in theory and practice. For piecewise constant dynamics (PCD), the continuous successor states (a.k.a. flow pipe) can be computed exactly, and the complexity is exponential in the number of variables [61; 63]. While in principle, any dynamics can be approximated arbitrarily well by PCD systems using an approach called hybridization [65], this requires partitioning of the state space, which often leads to prohibitive computational costs. For piecewise affine dynamics (PWA), one-step successors can be computed approximately using complex set representations. However, all published approaches suffer either from a possibly exponential increase in the complexity of the set representation, or from a possibly exponential increase in the approximation error as the considered time interval increases; this will be argued in detail in Sect. 6.3.

In addition to these theoretical obstacles, we note the following practical obstacles for the available tools and their performance in experiments. The only available model checkers that are (i) *sound* (i.e., they compute provable dense-time overapproximations), (ii) *unbounded* (i.e., they overapproximate the flowpipe for an infinite time horizon), and (iii) *arbitrarily precise* (i.e., they support precision refinement) are, with one exception, limited to PCD systems, namely, HyTech [64], PHAVer [52], and Lyse [23]. The tool Ariadne [20] can deal with affine dynamics and is sound, unbounded, and precise. However, Ariadne discretizes the reachable state space with a rectangular grid. This invariably leads to an exponential complexity in terms of the number of variables. Other tools that are applicable to PWA systems do not meet our criteria in that they are either not formally sound (e.g., CORA [4], SpaceEx [57]), not arbitrarily precise because of templates or particular data structures (e.g., SpaceEx, Flow* [35], CORA), or limited to bounded model checking (e.g., dReach [81], Flow*). All the above tools exhibit fatal limitations in scalability or precision on standard PWA benchmarks; they typically work only on well-chosen examples. Note that while these tools do not meet the criteria we advance in this paper, they of course have strengths in other areas handling nonlinear and nondeterministic dynamics.

**Our approach** We view iterative abstraction refinement as critical for soundness and precision management, and fixpoint detection as critical for evaluating unbounded properties. We implement, for the first time, a CEGAR (counterexample-guided abstraction refinement) scheme in combination with a fixpoint detection criterion for PWA systems. Our abstraction refinement scheme manages complexity and precision trade-offs in a flexible way by decoupling time from space: the dense timeline is partitioned into a sequence of intervals that are refined individually and lazily, by splitting intervals, to achieve the necessary precision and detect fixpoints; state sets are overapproximated using template polyhedra that are also refined individually and lazily, by adding normal directions to templates; and both refinement processes are interleaved for optimal results, while maintaining soundness with each step. A similar approach was recently proposed for the limited class of PCA systems [23]; this paper can be seen as an extension of the approach to the class of piecewise affine dynamics.

With each iteration of the CEGAR loop, a spurious counterexample is removed by computing a proof of infeasibility in terms of a sequence of linear constraints in space and interval constraints in time, which we call a sequence of *space-time interpolants*. We use linear programming to construct a suitable sequence of space-time intervals and check for fixpoints. If a fixpoint check fails, we increase the time horizon by adding new intervals. The separation of time from space gives us the flexibility to explore different refinement strategies. Fine-tuning the iteration of space refinement (adding template directions), time refinement (splitting intervals), and fixpoint checking (adding intervals), we find that it is generally best to prefer fewer time intervals over fewer space constraints. Based on performance evaluation, we even expand individual intervals time when this is possible without sacrificing the necessary precision for removing a counterexample.

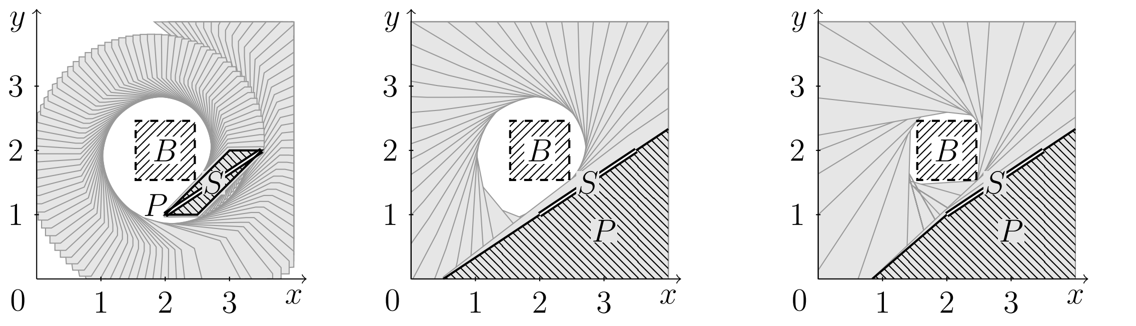
## 6.1 Motivating Example

The ordinary differential equation over the variables $x$ and $y$

$$\begin{aligned} \dot{x} &= 0.1x - y + 1.8 \\ \dot{y} &= x + 0.1y - 2.2 \end{aligned} \tag{6.1}$$

moves counterclockwise around the point $(2, 2)$ in an outward spiral. We center a box $B$ (of side 0.92) on the same point and place a diagonal segment $S$ close to the bottom right corner of $B$, without touching it (between $(2, 1)$ and $(3.5, 2)$; see Fig. 6.1). Then, we consider the problem of proving that every trajectory starting from any point in $S$ never hits $B$. This is a time-unbounded reachability problem for a hybrid automaton with piecewise affine dynamics and two control modes. The first mode has the dynamics above (Eq. 6.1) and $S$ as initial region. It has a transition to a second mode, which in its turn has $B$ as invariant. The second mode is a bad mode, which all trajectories indeed avoid.

We tackle the reachability problem by abstraction refinement. In particular, we aim at automatically constructing an enclosure for the flowpipe —i.e., for the set of trajectories from $S$— which (i) avoids the bad state $B$ and (ii) covers the continuous timeline up to infinity. Figure 6.1 shows three abstractions that result from different strategies for refining an initial space partition (i.e., template) and time partition (i.e., sequence of time intervals). All three refinement schemes start by enclosing $S$ with an initial template polyhedron $P$, and then transforming $P$ into a sequence of abstract flowpipe sections



Time refinement only (a) Uniform-time interpolation (b) Adaptive-time interpolation (c)

**Figure 6.1: Comparison of abstraction refinement methods**

$flow^{[\underline{t},\overline{t}]}(P)$, one for each interval $[\underline{t},\overline{t}]$ of an initial partitioning of the unbounded timeline. The computation of new flowpipe sections stops when a fixpoint is reached, —i.e., we reach a time threshold $t^*$ whose flowpipe section closes a cycle with $flow^{t^*}(P) \subseteq P$, sufficient condition for any further flowpipe section to be contained within the union of previously computed sections.

Refinement scheme (a) sticks to a fixed octagonal template $P$ —i.e., to the normals of a regular octagon— and iteratively halves all time intervals until every flowpipe section avoids the bad set $B$. This is achieved at interval width 1/64, but the computation does not terminate because no fixpoint is reached. Refinement scheme (b) splits time similarly but also computes a different, more accurate template for every iteration: first, an interval $[\underline{t},\overline{t}]$ is halved until it admits a halfspace interpolant —i.e., a halfspace $H$ that $S \subseteq H$ and $flow^{[\underline{t},\overline{t}]}(H) \cap B = \emptyset$; then, a maximal set of linearly independent directions is chosen as template from the normals of the obtained halfspaces. Refinement scheme (b) succeeds at interval width 1/16 to avoid $B$ and reach a fixpoint; the latter at time 6.25, with $flow^{6.25}(P) \subseteq P$. Refinement scheme (c) modifies (b) by optimizing the refinement of the time partition: instead of halving time intervals, the maximal intervals which admit halfspace interpolants are chosen. This scheme produces a nonuniform time partitioning with an average interval width of about 1/8, discovers five template directions, and finds a fixpoint in fewer steps.

Each iteration of the abstraction refinement loop consists of first abstracting the initial region into a template polyhedron, second solving the differential equation into a sequence of interval matrices, and finally transforming the template polyhedron using each of the interval matrices. We represent each transformation symbolically, by means of its support function. Then, we verify (i) the separation between every support function and the bad region, and (ii) the containment of any support function in the initial template polyhedron. The separation problem amounts to solving one LP, and the inclusion problem amounts to solving an LP in each template direction. If the separation fails, then we independently bisect each time that does not admit halfspace interpolants and expand each that does, until all are proven separated. Together, these halfspace interpolants form an infeasibility proof for the counterexample: a space-time interpolant. We forward the resulting new time intervals and halfspaces to the abstraction generator, and repeat, using the refined partitioning and the augmented template. If the inclusion fails, then we increase the time horizon by some amount $\Delta$, and repeat. Once we succeed with both separation and inclusion, the system is proved safe.

This example shows the advantage of lazily refining *both* the space partitioning (i.e., the template) by adding directions, and the time partitioning, by splitting intervals.

## 6.2 Hybrid Automata with Piecewise Affine Dynamics

A hybrid automaton with piecewise affine dynamics consists of an $n$-dimensional vector $x$ of real-valued variables and a finite directed multigraph $(V, E)$, the control graph. We call it the control graph, the vertices $v \in V$ the control modes, and the edges $e \in E$ the control switches. We decorate each mode $v \in V$ with an initial condition $Z_v \subseteq \mathbb{R}^n$, a nonnegative invariant condition $I_v \subseteq \mathbb{R}^n_{\geq 0}$, and a flow condition given by the system of ordinary differential equations

$$\dot{x} = A_v x + b_v. \tag{6.2}$$

We decorate each switch $e \in E$ with a guard condition $G_e \subseteq \mathbb{R}^n$ and an update condition given the difference equations $x := R_e x + s_e$. All constraints $I$, $G$, and $Z$ are conjuctions of rational linear inequalities, $A$ and $R$ are constant matrices, and $b$ and $s$ constant vectors of rational coefficients. In this paper, whenever an indexing of modes and switches is clear from the context, we index the respective constraints and transformations similarly, e.g., we abbreviate $A_{v_i}$ with $A_i$.

A trajectory is a possibly infinite sequence of states $(v, x) \in V \times \mathbb{R}^n$ repeatedly interleaved first by a switching time $t \in \mathbb{R}_{\geq 0}$ and then by a switch $e \in E$

$$(v_0, x_0)t_0(v_0, y_0)e_0(v_1, x_1)t_1(v_1, y_1)e_1 \dots \tag{6.3}$$

for which there exists a sequence of solutions $\psi_0, \psi_1, \dots : \mathbb{R} \to \mathbb{R}_n$ such that $\psi_i(0) = x_i$, $\psi_i(t_i) = y_i$ and they satisfy (i) the invariant conditions $\psi_i(t) \in I_i$ and (ii) the flow conditions $\dot{\psi}_i(t) = A_i\psi_i(t) + b_i$, for all $t \in [0, t_i]$. Moreover, $x_0 \in Z_0$, every switch $e_i$ has source $v_i$, destination $v_{i+1}$, and and the respective states satisfy (i) the guard condition $y_i \in G_i$ and (ii) the update $x_{i+1} = R_i y_i + s_i$. The maximal set of its trajectories is the semantics of the hybrid automaton, which is safe if none of them contains a special bad mode.

Every hybrid automaton with affine dynamics can be transformed into an equivalent hybrid automaton with linear dynamics, i.e., the special case where $b = 0$ on every mode. We obtain such transformation by adding one extra variable $y$, rewriting the flow of every mode into $\dot{x} = Ax + by$, and forcing $y$ to be always equal to 1, i.e., invariant $y = 1$ and flow $\dot{y} = 0$ on every mode and update $y' = y$ on every switch. For this reason, in the following sections we discuss w.l.o.g. the reachability analysis of hybrid automata with linear dynamics.

## 6.3 Time Abstraction using Interval Arithmetic

We abstract the reach set of the hybrid automaton with a union of convex polyhedra. In particular, we abstract the states that are reachable in a mode using a finite sequence of images of the initial region over a *time partitioning*, until a completeness threshold is reached. Thereafter, we compute the *template polyhedron* of each of the images that can take a switch. Then, we repeat in the destination mode and we continue until a fixpoint is found.

Precisely, a time partitioning $T$ is a (possibly infinite) set of disjoint closed time intervals whose union is a single (possibly open) interval. For a finite set of directions $D \subseteq \mathbb{R}^n$, the $D$-polyhedron of a closed convex set $X$ is the tightest polyhedral enclosure whose facets normals are in $D$. In the following, we associate every mode $v$ to a template $D_v$ and a time partitioning $T_v$ of the time axis $\mathbb{R}_{\geq 0}$, we employ interval arithmetic for abstracting the continuous dynamics (Sec. 6.3.1), and on top of it we develop a procedure for hybrid dynamics (Sec. 6.3.2).

## 6.3.1 Continuous Dynamics

We consider w.l.o.g. a mode with ODE reduced to the linear form $\dot{x} = A_v x$, invariant $I_v$, and a given time interval $[\underline{t}, \overline{t}]$. Every linear ODE $\dot{x} = Ax$ has the unique solution

$$\psi(t) = \exp(At)\psi(0). \tag{6.4}$$

It follows (see also [59]) that the set of states reachable in $v$ after exactly $t$ time units from an initial region $X$ is

$$\text{flow}_v^t(X) = \exp(A_v t)X \cap \bigcap_{0 \leq \tau \leq t} \exp\big(A_v(t-\tau)\big)I_v, \tag{6.5}$$

Then, the flowpipe section over the time interval $[\underline{t}, \overline{t}]$ is

$$\text{flow}_v^{[\underline{t},\overline{t}]}(X) = \cup\{\text{flow}_v^t(X) \mid t \in [\underline{t}, \overline{t}]\}. \tag{6.6}$$

We note three straightforward but consequential properties of the reach set: (i) The accuracy of any convex abstraction depends on the size of the time interval: While $\text{flow}_v^t(X)$ is convex for convex $X$, this is generally not the case for $\text{flow}_v^{[\underline{t},\overline{t}]}(X)$. (ii) We can prune the time interval whenever we detect that the reach set no longer overlaps with the invariant: If for any $t^* \geq 0$, $\text{flow}_v^{t^*}(X) = \emptyset$, then for all $\overline{t} \geq t^*$, $\text{flow}_v^{\overline{t}}(X) = \emptyset$ and $\text{flow}_v^{[\underline{t},\overline{t}]}(X) = \text{flow}_v^{[\underline{t},t^*]}(X)$. (iii) We can prune the time interval whenever we detect containment in the initial states: If $\text{flow}_v^{t^*}(X) \subseteq X$, then $\text{flow}_v^{[\underline{t},\infty]}(X) = \text{flow}_v^{[\underline{t},t^*]}(X)$.

For given $A$ and $t$, the matrix $\exp(At)$ can be computed with arbitrary, but only finite, accuracy. We resolve this problem by computing a rational interval matrix $[\underline{M}, \overline{M}]$, which we denote $\text{intexp}(A, \underline{t}, \overline{t})$, such that for all $t \in [\underline{t}, \overline{t}]$ we have element-wise that

$$\exp(At) \in \text{intexp}(A, \underline{t}, \overline{t}). \tag{6.7}$$

This interval matrix can be derived efficiently with a variety of methods [89], e.g., using a guaranteed ODE solver or using interval arithmetic. The width of the interval matrix can be made arbitrarily small at the price of increasing the number of computations and the size of the representation of the rational numbers. In our approach, we do not rely in a fixed accuracy of the interval matrix. Instead, we require that the accuracy increases as the width of the time interval goes to zero. That way, we don't need to introduce an extra parameter. To ensure progress in our refinement loop, we require that the interval matrix decreases monotonically when we split the time interval. Formally, if $[\underline{t}, \overline{t}] \subseteq [\underline{u}, \overline{u}]$ we require the element-wise inclusion $\mathrm{intexp}(A, \underline{t}, \overline{t}) \subseteq \mathrm{intexp}(A, \underline{u}, \overline{u})$. This can be ensured by intersecting the interval matrices with the original interval matrix after time splitting.

While the mapping with interval matrices is in general not convex [100], we can simplify the problem by assuming that all points of $X$ are in the positive orthant. As long as $X$ is bounded from below, this condition can be satisfied by inducing an appropriate coordinate change. Under the assumption that $X \subseteq \mathbb{R}^n_{\geq 0}$,

$$[\underline{M}, \overline{M}](X) = \big\{ y \in \mathbb{R}^n \;\big|\; \underline{M}x \leq y \leq \overline{M}x \text{ and } x \in X \big\}. \tag{6.8}$$

Combining the above results, we obtain a convex abstraction of the flowpipe over a time interval as

$$\mathit{flow}_v^{[\underline{t},\overline{t}]}(X) = \mathrm{intexp}(A, \underline{t}, \overline{t})X \cap I_v. \tag{6.9}$$

The abstraction is conservative in the sense that $\mathrm{flow}_v^{[\underline{t},\overline{t}]}(X) \subseteq \mathit{flow}_v^{[\underline{t},\overline{t}]}(X)$. On the other hand, the longer is the time interval, the coarser is the abstraction. For this reason, we construct an abstraction of the flowpipe in terms of a union of convex approximations over a time partitioning. The abstract flowpipe over the time partitioning $T$ is

$$\mathit{flow}_v^T(X) = \cup\{\mathit{flow}_v^{[\underline{t},\overline{t}]}(X) \mid [\underline{t}, \overline{t}] \in T\}. \tag{6.10}$$

Again, this is conservative w.r.t. the concrete flowpipe, i.e., for all time partitionings $T$ it holds that $\mathrm{flow}_v^{\cup T}(X) \subseteq \mathit{flow}_v^T(X)$. Moreover, it is conservative w.r.t. any refinement of $T$, i.e., the time partitioning $U$ refines $T$ if $\cup U = \cup T$ and $\forall [\underline{u}, \overline{u}] \in U \colon \exists [\underline{t}, \overline{t}] \in T \colon [\underline{u}, \overline{u}] \subseteq [\underline{t}, \overline{t}]$, then $\mathit{flow}_v^U(X) \subseteq \mathit{flow}_v^T(X)$.

### 6.3.2 Hybrid Dynamics

We embed the flowpipe abstraction routine into a reachability algorithm that accounts for the switching induced by the hybrid automaton. The discrete post operator is the image of a set $Y \subseteq \mathbb{R}^n$ through a switch $e \in E$

$$jump_e(Y) = R_e(Y \cap G_e) \oplus \{s_e\}. \tag{6.11}$$

We explore the hybrid automaton constructing a set of abstract trajectories, namely sequences abstract states interleaved by time intervals and switches

$$(v_0, X_0)[\underline{t}_0, \overline{t}_0](v_0, Y_0)e_0(v_1, X_1)[\underline{t}_1, \overline{t}_1](v_1, Y_1)e_1 \dots \tag{6.12}$$

where $X_0, Y_0, \dots \subseteq \mathbb{R}^n$ are nonempty sets of states that comply with template $\{D_v\}$ and partitioning $\{T_v\}$ in the following sense. First, $X_0 = Z_0$ and $X_{i+1} = jump_i(Y_i)$ for all $i \geq 0$. Second, $Y_i = flow_i^{[\underline{t}_i, \overline{t}_i]}(P_i)$ for all $i \geq 0$, where $P_i$ is the $D_i$-polyhedron of $X_i$ and $[\underline{t}_i, \overline{t}_i] \in T_i$. The maximal set of abstract trajectories, the abstract semantics induced by $\{D_v\}$ and $\{T_v\}$, overapproximates the concrete semantics in the sense that every concrete trajectory (see Eq. 6.3) has an abstract trajectory that subsumes it, i.e., modes and switches match, $x_i \in X_i$, $t_i \in [\underline{t}_i, \overline{t}_i]$, and $y_i \in Y_i$, for all $i \geq 0$.

Computing the abstraction involves several difficulties. First, the trajectories might be not finitary. Indeed, this is unsolvable in theory, because the reachability problem is undecidable [71]. Second, the post operators are hard to compute. In particular, obtaining the sets $X$ and $Y$ in terms of conjunctions of linear inequalities in $\mathbb{R}^n$ requires eliminating quantifiers. In Alg. 2, we present a procedure (which does not necessarily terminate) for tackling the first problem. In the next section, we show how to tackle the second using support functions.

We employ Alg. 2 to explore the tree of abstract trajectories. We store in the stack $W$ the leaves to process $\dots(v, X)$, followed by a candidate interval $[\underline{t}, \overline{t}]$. For each leaf, we retrieve $P$, the template polyhedron of $X$. If it leads to a bad mode, we return, otherwise we search for a completeness threshold $t^*$ between $\underline{t}$ excluded and $\overline{t}$, checking for inclusion in the union of visited polyhedra $P_v$. In case of failure, we extend the time horizon of $\Delta$ and push the next candidate to the stack. Then, we partition the time between $\underline{t}$ and $t^*$, construct the flowpipe, and process switching. Upon each successful switch, we augment $P_{v'}$ with the $D_{v'}$-polyhedron of the switching region $X'$, avoiding to store redundant

**Algorithm 2:** Reachability procedure.

```
input  : Template $\{D_v\}$ and partitioning $\{T_v\}$ indexed by $V$
output : Optionally an abstract trajectory (counterexample)
foreach $v \in V$ with nonempty $Z_v$ do
    push $(v, Z_v)[0, \Delta]$ into the stack $W$;
    add the $D_v$-polyhedron of $Z_v$ to $P_v$;
end
while $W$ is not empty do
    pop $\dots(v, X)[\underline{t}, \overline{t}]$ from $W$;
    $P \leftarrow D_v$-polyhedron of $X$;
    if $v$ is bad and $P \cap I_v$ is nonempty then                 // check counterexample
        return $\dots(v, X)$;
    end
    foreach $t^* \in \{\underline{t} + \delta, \underline{t} + 2\delta, \dots, \overline{t}\}$ do          // find completeness threshold
        if $flow_v^{t^*}(P) \subseteq P_v$ then break;
    end
    if $t^* = \overline{t}$ and $flow_v^{\overline{t}}(P) \not\subseteq P_v$ then              // otherwise extend time horizon
        push $\dots(v, X)[\overline{t}, \overline{t} + \Delta]$ into $W$;
    end
    foreach $[\underline{u}, \overline{u}] \in T_v$ and $[\underline{u}, \overline{u}] \cap [\underline{t}, t^*] \neq \emptyset$ do         // construct flowpipe
        $Y \leftarrow flow_v^{[\underline{u}, \overline{u}]}(P)$;
        foreach $e \in E$ with source $v$ and destination $v'$ do
            $X' \leftarrow jump_e(Y)$;
            if $X' \subseteq P_{v'}$ then continue;
            push $\dots(v, X)[\underline{u}, \overline{u}](v, Y)e(v', X')[0, \Delta]$ into $W$;
            add the $D_{v'}$-polyhedron of $X'$ to $P_{v'}$;
        end
    end
end
```

polyhedra. Notably, the latter operation is efficient because all polyhedra comply with the same template. For the same reason, we obtain efficient inclusion checks, which we implement by first computing the template polyhedron of the left hand side, and then comparing the constant terms of the respective linear inequalities.

In conclusion, this reachability procedure that takes a template $\{D_v\}$ and a partitioning $\{T_v\}$ and constructs a tree of reachable sets of states $X$ and $Y$. It manipulates them through the post operators and overapproximate them into template polyhedra. In the next section, we discuss how to efficiently represent $X$ and $Y$, so to efficiently compute their template polyhedra. In Sec. 6.5 we discuss how to discover appropriate $\{D_v\}$ and $\{T_v\}$, so to eliminate spurious counterexamples.

## 6.4 Space Abstraction using Support Functions

Abstracting away time left us with the task of representing the state space of the hybrid automaton, namely the space of its variable valuations. Such sets consists of polyhedra emerging from operations such as intersections, Minkowski sums, and linear maps with simple or interval matrices. In this section, we discuss how to represent precisely all sets emerging from any of these operations by means of their support functions (Sec. 6.4.1) and then how to abstract them into template polyhedra (Sec. 6.4.2). In the next section, we discuss how to refine the abstraction.

### 6.4.1 Support Functions

The support function of a closed convex set $X \subseteq \mathbb{R}^n$ in direction $d \in \mathbb{R}^n$ consists of the maximizer scalar product of $d$ over $X$

$$\rho_X(d) = \sup\{d^{\mathsf{T}} x \mid x \in X\}, \tag{6.13}$$

and, indeed, uniquely represents any closed convex set [99]. Classic work on the verification of hybrid automata with affine dynamic have posed a framework for the construction of support functions from basic set operations, but under the assumption of unboundedness and nonemptiness of the represented set, and with approximated intersection [59]. Indeed, if the set is empty then its support function is $-\infty$, while if it is unbounded an $d$ points toward a direction of recession is $+\infty$, making the framework end up into undefined values. Such conditions turn out to be limiting in our context, first because we find desirable to represent unbounded sets so to accelerate the convergence to a fixpoint of the abstraction procedure, but most importantly because when encoding support functions for long abstract trajectories we might be not aware whether its concretization is infeasible. Checking this is a crucial element of a counterexample-guided abstraction refinement routine.

Recent work on the verification of hybrid automata with constant dynamics, i.e., with flows defined by constraints on the derivative only, provides us with a generalization of the classic support function framework which relaxes away the assumptions of boundedness and nonemptiness and yields precise intersection [23]. The framework encodes combinations of convex sets of states into LP (linear programs) which enjoy strong duality with their support function. Similarly, we encode the support function in direction $d$ of any set $X$ into the LP

$$\begin{aligned} \text{minimize} \quad & c^{\mathsf{T}}\lambda \\ \text{subject to} \quad & A\lambda = Bd, \end{aligned} \tag{6.14}$$

over the nonnegative vector of variables $\lambda$. The LP is dual to $\rho_X(d)$, which is to say that if the LP is infeasible then $X$ is unbounded in direction $d$, and if the LP is unbounded then $X$ is the empty set. Moreover, if the LP has bounded solution so does $\rho_X(d)$ and the solutions coincide.

The construction is inductive on operations between sets. For the base case, we recall that from duality of linear programming the support function of a polyhedron given by a system of inequalities $Px \leq q$ is dual to the LP over $\lambda \geq 0$

$$\begin{aligned} \text{minimize} \quad & q^{\mathsf{T}}\lambda \\ \text{subject to} \quad & P^{\mathsf{T}}\lambda = d. \end{aligned} \tag{6.15}$$

Then, inductively, we assume that for the set $X \subseteq \mathbb{R}^n$ we are given an LP with the coefficients $A_X$, $B_X$, and $c_X$, and similarly for the set $Y \subseteq \mathbb{R}^n$. For the support functions of $X \oplus Y$, $MX$, and $X \cap Y$ we respectively construct the following LP over the nonnegative vectors of variables $\lambda$, $\mu$, $\alpha$, and $\beta$:

$$\begin{aligned} \text{minimize} \quad & c_X^{\mathsf{T}}\lambda + c_Y^{\mathsf{T}}\mu \\ \text{subject to} \quad & A_X\lambda = B_Xd \text{ and } A_Y\mu = B_Yd, \end{aligned} \tag{6.16}$$

$$\begin{aligned} \text{minimize} \quad & c_X^{\mathsf{T}}\lambda \\ \text{subject to} \quad & A_X\lambda = B_XM^Td, \text{ and} \end{aligned} \tag{6.17}$$

$$\begin{aligned} \text{minimize} \quad & c_X^{\mathsf{T}}\lambda + c_Y^{\mathsf{T}}\mu \\ \text{subject to} \quad & A_X\lambda - B_X(\alpha - \beta) = 0 \text{ and} \\ & A_Y\mu + B_Y(\alpha - \beta) = B_Yd. \end{aligned} \tag{6.18}$$

Such construction follows as a special case of [23], which we extend with the support function of a map through an interval matrix.

The time abstraction of Sec. 6.3 additionally requires us to represent the map of sets of states through interval matrices. Precisely, we are given convex set of nonnegative values $X \subseteq \mathbb{R}^n_{\geq 0}$, the coefficients for the respective LP, an interval matrix $[\underline{M}, \overline{M}] \subseteq \mathbb{R}^{n \times n}$, and we aim at computing the support function of all values in $X$ mapped by all matrices in

$[\underline{M}, \overline{M}]$. To this end, we define the LP

$$\begin{aligned} &\text{minimize} && c_X^\mathsf{T} \lambda \\ &\text{subject to} && A_X \lambda + B_X(\underline{M}^\mathsf{T} \mu - \overline{M}^\mathsf{T} \nu) = 0 \text{ and} \\ & && -\mu + \nu = d, \end{aligned} \tag{6.19}$$

over the vectors $\lambda$, $\mu$, and $\nu$ of nonnegative variables. This linear program corresponds to the the dual of the interval matrix map in Eq. 6.8.

### 6.4.2 Computing Template Polyhedra

We represent all space abstractions $X$ and $Y$ in our procedure by their support functions. In particular, whenever set operations are applied, instead of solving the operation by removing quantifiers, we construct an LP. We delay solving it until we need to compute a template polyhedron. In that case, we compute the $D$-polyhedron of the set $X$ by computing its support function in each of the directions in $D$, and constructing the intersection of halfspaces $\cap\{d^\mathsf{T} x \leq \rho_X(d) \mid d \in D\}$.

## 6.5 Abstraction Refinement using Space-time Interpolants

The reachability analysis of hybrid automata by means of the combination of interval arithmetic and support functions presented in Sec. 6.3 and 6.4 builds an overapproximation of the system dynamics. It is always sound for safety, but it may produce spurious counterexamples, due to an inherent lack of precision of the time abstraction and the polyhedral approximation. The level of precision is given by two factors, namely the choice of time partitioning and the choice of template directions, excluding the parameters for approximation of the exponential function, which we assume constant (see Sec. 6.3.1). In the following, we present a procedure to extract infeasibility proofs from spurious counterexamples. We produce them in the form of time partitions and bounding polyhedra, which we call space-time interpolants. Space-time interpolants can then be used to properly refine time partitioning and template directions.

Consider the bounded path $v_0, e_0, v_1, e_1, \ldots, v_k, e_k, v_{k+1}$ over the control graph and a sequence of dwell time intervals $[\underline{t}_0, \overline{t}_0], [\underline{t}_1, \overline{t}_1], \ldots, [\underline{t}_k, \overline{t}_k]$ emerging from an abstract trajectory. We aim at extracting a sequence $X_0, X_1, \ldots, X_{k+1}$ of (possibly nonconvex) polyhedra and a sequence $T_0, T_1, \ldots, T_k$ of refinements of the respective dwell times such that $Z_0 \subseteq X_0$, $jump_0 \circ flow_0^{T_0}(X_0) \subseteq X_1$, $\ldots$, $jump_k \circ flow_k^{T_k}(X_k) \subseteq X_{k+1}$, and $X_{k+1} \cap I_{k+1}$ is empty. In other words, we want every $X_{i+1}$ to contain all states that can enter mode $v_{i+1}$ after dwelling on $v_i$ between $\underline{t}_i$ and $\overline{t}_i$ time, and the last to be separated from the invariant of mode $v_{k+1}$. Containment is to hold inductively, namely $X_{i+1}$ has to contain what is reachable from $X_i$, and the time refinements $T$ are to be chosen in such a way that containment holds in the abstraction. Then, we call the sequence $X_0, T_0, X_1, T_1, \ldots, X_k, T_k, X_{k+1}$ a sequence of space-time interpolants for the path and the dwell times above.

We compute a sequence of space-time interpolants by alternating multiple strategies. First, for the given sequence of dwell times, we attempt to extract a sequence of halfspace interpolants using linear programming (Sec. 6.5.1). In case of failure, we iteratively partition the dwell times in sets of smaller intervals, separating nonswitching from switching times and until every combination of intervals along the path admits halfspace interpolants (Sec. 6.5.2). We accumulate all halfspaces to form a sequence of unions of convex polyhedra that, together with the obtained time partitionings, will form a valid sequence of space-time interpolants. Finally, we refine the abstraction using the time partitionings and the outwards pointing directions of all computed halfspaces, in order to eliminate the spurious counterexample (Sec. 6.5.3).

### 6.5.1 Halfspace Interpolation

Halfspace interpolants are the special case of space-time interpolants where every polyhedron in the sequence is defined by a single linear inequality [2]. Indeed, they are the simplest kind of space-time interpolants, and, for the same reason, the ones that best generalize the reachable states along the path. Unfortunately, not all paths admit halfspace interpolants, but, if one such sequence exists, then it can be extrapolated from the solution of a linear program.

Consider a path $v_0, e_0, \dots, v_{k+1}$ with the respective dwell times $[\underline{t}_0, \overline{t}_0], \dots, [\underline{t}_k, \overline{t}_k]$. A sequence of halfspace interpolants consists of a sequence of sets $H_0, \dots, H_{k+1}$ among either any halfspace, or the empty set, or the universe, such that $Z_0 \subseteq H_0$, $jump_0 \circ flow_0^{[\underline{t}_0, \overline{t}_0]}(H_0) \subseteq H_1$, $\dots$, $jump_k \circ flow_k^{[\underline{t}_k, \overline{t}_k]}(H_k) \subseteq H_{k+1}$, and $H_{k+1} \cap I_{k+1}$ is empty. In contrast with general space-time interpolants, every time partition consists of a single time interval and therefore the support function of every post operator $jump \circ flow^{[\underline{t}, \overline{t}]}$ can be encoded into a single LP (see Sec. 6.4). We exploit the encoding for extracting halfspace interpolants, similarly to a recent interpolation technique for PCD systems [23].

We encode the support function in direction $d$ of the closure of the image of the post operators along the path, i.e., the set $jump_k \circ flow_k^{[\underline{t}_k, \overline{t}_k]} \circ \dots \circ jump_0 \circ flow_0^{[\underline{t}_0, \overline{t}_0]}(Z_0)$, intersected with the invariant $I_{k+1}$. We obtain the following LP over the free vectors $\alpha_0, \dots, \alpha_{k+1}$ and the nonnegative vectors $\beta$, $\delta_0, \dots, \delta_k$, $\gamma_0, \dots, \gamma_{k+1}$, $\mu_0, \dots, \mu_k$, and $\nu_0, \dots, \nu_k$:

$$\begin{array}{lllr}
\text{minimize} & q_{Z_0}^\mathsf{T}\beta + \sum_{i=0}^{k}(q_{I_i}^\mathsf{T}\gamma_i + q_{G_i}^\mathsf{T}\delta_i + s_i^\mathsf{T}\alpha_{i+1}) + q_{I_{k+1}}^\mathsf{T}\gamma_{k+1} & & \\
\text{subject to} & P_{Z_0}^\mathsf{T}\beta = \alpha_0, & & \\
& \underline{M}_i^\mathsf{T}\mu_i - \overline{M}_i^\mathsf{T}\nu_i = -\alpha_i & \text{for each } i \in [0..k], & (6.20) \\
& -\mu_i + \nu_i + P_{I_i}^\mathsf{T}\gamma_i + P_{G_i}^\mathsf{T}\delta_i = R_i^\mathsf{T}\alpha_{i+1} & \text{for each } i \in [0..k], & \\
& P_{I_{k+1}}^\mathsf{T}\gamma_{k+1} = -\alpha_{k+1} + d, & &
\end{array}$$

where every system of inequalities $Px \leq q$ corresponds to the constraints of the respective init, guard, or invariant, every $R_i x + s_i$ is an update equation, and every interval matrix $[\overline{M}_i, \underline{M}_i] = \text{intexp}(A_i, \underline{t}_i, \overline{t}_i)$. In general, one can check whether the closure is contained in a halfspace $a^\mathsf{T}x \leq b$ by setting the direction to its linear term $d = a$ and checking whether the objective function can equal its constant term $b$. In particular, we check for emptiness, which we pose as checking inclusion in $0x \leq -1$. Therefore, we set $d = 0$ and the objective function to equal $-1$. Upon affirmative answer, from the solution $\alpha_0^\star, \alpha_1^\star, \dots, \nu_k^\star$ we obtain a valid sequence of halfspace interpolants whose $i$-th linear term is given by $\alpha_i^\star$ and $i$-th constant term is given by $q_{Z_0}^\mathsf{T}\beta^\star + \sum_{j=0}^{i-1}(q_{I_j}^\mathsf{T}\gamma_j^\star + q_{G_j}^\mathsf{T}\delta_j^\star + s_j^\mathsf{T}\alpha_{j+1}^\star)$.

## 6.5.2 Time Partitioning

Halfspace interpolation attempts to compute a sequence of enclosures that are convex for a sequence of sets that are not necessarily convex. Specifically, it requires each halfspace to enclose the set of solutions of a linear differential equation, which is nonconvex, by enclosing its convex overapproximation along a whole time interval. As a result, large

**Algorithm 3:** Nonswitching time partitioning.

```
input  : sequence of intervals $[\underline{u}_0, \overline{u}_0], \dots, [\underline{u}_j, \overline{u}_j]$
output : set of intervals
$b \leftarrow \underline{u}_j$;
while $b < \overline{u}_j$ do
    $a \leftarrow b$;
    $b \leftarrow b + \varepsilon$ ;
    $c \leftarrow \overline{u}_j$;
    if $[\underline{u}_0, \overline{u}_0], \dots, [\underline{u}_{j-1}, \overline{u}_{j-1}], [a, b]$ does not admit halfspace interpolants then
        continue;
    end
    if $[\underline{u}_0, \overline{u}_0], \dots, [\underline{u}_{j-1}, \overline{u}_{j-1}], [a, c]$ admits halfspace interpolants then
        push $[a, c]$ to the output;
        return;
    end
    while $c - b > \varepsilon$ do
        if $[\underline{u}_0, \overline{u}_0], \dots, [\underline{u}_{j-1}, \overline{u}_{j-1}], [a, \varepsilon\lfloor \frac{b+c}{2\varepsilon} \rfloor]$ admits halfspace interpolants then
            $b \leftarrow \varepsilon\lfloor \frac{b+c}{2\varepsilon} \rfloor$;
        else
            $c \leftarrow \varepsilon\lfloor \frac{b+c}{2\varepsilon} \rfloor$;
        end
    end
    push $[a, b]$ to the output;
end
```

time intervals produce large overapproximations, on which halfspace interpolation might be impossible. Likewise, shorter intervals produce tighter overapproximations, which are more likely to admit halfspace interpolants. In this section, we exploit such observation to enable interpolation over large time intervals. In particular, we properly partition the time into smaller subintervals and we treat each of them as a halfspace interpolation problem. Later, we combine the results to refine the abstraction.

Time partitioning is a delicate task in the whole abstraction refinement loop. In fact, while template refinement affects linearly the performance of the abstractor, partitioning time intervals that can switch induces branching in the search, possibly leading to an exponential blowup. For this reason, we partition time by narrowing down the switching time, for incremental precision, until no more is left. In particular, we use Alg. 3 to compute a set $N$ of maximal intervals that admit halfspace interpolants, by enlarging or narrowing them of $\varepsilon$ amounts. We embed this procedure in Alg. 4 which, along the

**Algorithm 4:** Dwell time partitioning.

```
input  : sequence of intervals $[\underline{t}_0, \overline{t}_0], \dots, [\underline{t}_k, \overline{t}_k]$
output : set of sequences of intervals
1  push $[\underline{t}_0, \overline{t}_0]$ to the queue $Q$;
2  while $Q$ is not empty do
3      pop $[\underline{u}_0, \overline{u}_0], \dots, [\underline{u}_j, \overline{u}_j]$ from $Q$;
4      $N \leftarrow$ nonswitching time partitioning of $[\underline{u}_0, \overline{u}_0], \dots, [\underline{u}_j, \overline{u}_j]$;
5      foreach $[\underline{a}, \overline{a}] \in N$ do
6          push $[\underline{u}_0, \overline{u}_0], \dots, [\underline{u}_{j-1}, \overline{u}_{j-1}], [\underline{a}, \overline{a}]$ to the output;
7      end
8      if $j = k$ then
9          assert $[\underline{u}_j, \overline{u}_j] \setminus \cup N = \emptyset$;
10         continue;
11     end
12     $S \leftarrow$ choose set of intervals that cover $[\underline{u}_j, \overline{u}_j] \setminus \cup N$;
13     foreach $[\underline{b}, \overline{b}] \in S$ do
14         push $[\underline{u}_0, \overline{u}_0], \dots, [\underline{u}_{j-1}, \overline{u}_{j-1}], [\underline{b}, \overline{b}], [\underline{t}_{j+1}, \overline{t}_{j+1}]$ to $Q$;
15     end
16 end
```

sequence, excludes the time in $N$, constructing a set of intervals $S$ that overapproximate the switching time. In particular, we construct the set with the widest possible intervals that are disjoint from $N$. Algorithm 4 succeeds when no more intervals are left, otherwise we half $\varepsilon$ and reapply it to the sequences that are left to process.

### 6.5.3 Abstraction Refinement

The procedures above construct sequences of time intervals $[\underline{u}_0, \overline{u}_0], \dots, [\underline{u}_j, \overline{u}_j]$ that are included in $[\underline{t}_0, \overline{t}_0], \dots, [\underline{t}_k, \overline{t}_k]$ and that, with the respective halfspace interpolants, this constitutes a proof of infeasibility for the counterexample. Yet, it does not form a sequence of space-time interpolants $X_0, T_0, \dots, X_{k+1}$.We form each partitioning $T_i$ by splitting $[\underline{t}_i, \overline{t}_i]$ in such a way each element of $T_i$ is either contained in $[\underline{u}_i, \overline{u}_i]$ or disjoint from it, for all intervals $[\underline{u}_i, \overline{u}_i]$. Then, we refine the partitioning of mode $v_i$ similarly. Each polyhedron $X_i$ is a union of convex polyhedra, each of which is the intersection of all halfspaces $H_i$ corresponding to some sequence $[\underline{u}_0, \overline{u}_0], \dots, [\underline{u}_i, \overline{u}_i]$. Nevertheless, to refine the abstraction we do not need to construct $X_i$, but just to take the outward point directions of all $H_i$ and add them to the template of $v_i$.

| | # vars | # modes | # cex | # dirs | avg. width | cex. time | ref. time | ver. time | tot. time | Ariadne |
|---|---|---|---|---|---|---|---|---|---|---|
| filtosc_1st_ord | 3 | 4 | 7 | 13 | 0.55 | 0.57 | 0.96 | 0.13 | **1.66** | 27.56 |
| filtosc_2nd_ord | 4 | 4 | 7 | 15 | 0.55 | 0.83 | 1.78 | 0.20 | **2.81** | 150.7 |
| filtosc_3rd_ord | 5 | 4 | 7 | 16 | 0.55 | 1.28 | 4.65 | 0.32 | **6.25** | oot |
| filtosc_4th_ord | 6 | 4 | 7 | 18 | 0.55 | 1.53 | 11.39 | 0.37 | **13.29** | oot |
| filtosc_5th_ord | 7 | 4 | 7 | 19 | 0.55 | 2.61 | 26.60 | 0.70 | **29.37** | - |
| filtosc_6th_ord | 8 | 4 | 7 | 18 | 0.55 | 4.56 | 101.8 | 1.29 | **107.7** | - |
| filtosc_7th_ord | 9 | 4 | 7 | 18 | 0.55 | 4.36 | 109.9 | 1.13 | **114.6** | - |
| filtosc_8th_ord | 10 | 4 | 7 | 17 | 0.55 | 5.92 | 150.9 | 1.54 | **158.4** | - |
| filtosc_9th_ord | 11 | 4 | 7 | 16 | 0.55 | 6.49 | 383.1 | 1.83 | **391.3** | - |
| filtosc_10th_ord | 12 | 4 | 7 | 17 | 0.55 | 12.84 | 428.87 | 3.73 | **445.4** | - |
| filtosc_11th_ord | 13 | 4 | 7 | 17 | 0.55 | 15.10 | 525.2 | 4.38 | **544.6** | - |
| reactor_1_rod | 2 | 4 | 11 | 3 | 0.11 | 5.24 | 10.64 | 1.59 | **17.47** | oot |
| reactor_2_rods | 3 | 5 | 9 | 7 | 0.79 | 5.68 | 5.36 | 2.33 | **13.37** | oot |
| reactor_3_rods | 4 | 6 | 12 | 13 | 1.07 | 14.46 | 13.94 | 13.13 | **41.53** | - |
| reactor_4_rods | 5 | 7 | 15 | 29 | 1.67 | 45.50 | 42.47 | 111.5 | **199.9** | - |
| reactor_5_rods | 6 | 8 | 16 | 31 | 1.81 | 73.77 | 27.36 | 696.46 | **797.5** | - |

**Table 6.1: Statistics for the benchmark examples (oot when ¿ 1000s).**

## 6.6 Experimental Evaluation

We implemented our method in `C++` using GMP and Eigen for multiple precision linear algebra, Arb for interval arithmetic, and PPL for linear programming [76; 17]. In particular, all libraries we are using are meant to provide guaranteed solutions, as well as our implementation. We evaluate it on several instances of a *filtered oscillator* and a *rod reactor*, which are both parametric in the number of variables, and the latter in the number of modes too [57; 111]. We record several statistics from every execution of our tool: the number #cex of counterexamples found during the CEGAR loop, the number #dir of linearly independent directions and the average width of the time partitionings extracted from all space-time interpolants. Moreover, we independently measure three times. First, the time spent in finding counterexamples, namely the total time taken by inconclusive abstractions which returned a spurious counterexample. Second, the refinement time, that is the total time consumed by computing space-time interpolants. Finally, the verification time, that is the time spend in the last abstraction of the CEGAR loop, which terminates with a fixpoint proving the system safe. We compare the outcome and the performance of our tool against Ariadne which, to the best of our knowledge, is the only verification tool available that is numerically sound and time-unbounded [45].

The filtered oscillator is hybrid automaton with four modes that smoothens a signal $x$ into a signal $z$. It has $k + 2$ variables and a system of $k + 2$ affine ODE, where $k$ is the order of the filter. Table 6.1 shows the results, for a scaling of $k$ up to the 11-th order. The first observation is that the CEGAR loop behaves quite similarly on all scalings: number of counterexamples, number of directions, and time partitionings are almost identical. On the other hand, the computation times show a growth, particularly in the refinement phase which dominates over abstraction and verification. This suggests us that our procedure exploits efficiently the symmetries of the benchmark. In particular, time partitioning seems unaffected. What affects the performance is linear programming, whose size depends on the number of variables of the system.

The rod reactor consists of a heating reactor tank and $k$ rods each of which cools the tank for some amount of time, excluding each other. The hybrid automaton has one variable $x$ for the temperature, $k$ clock variables, one heating mode, one error mode, and $k$ cooling modes. If the temperature reaches a critical threshold and no rod can intervene, it goes into an error. For this benchmark, we start with a simple template, the interval around $x$, and we discover further directions. Table 6.1 highlights two fundamental differences with the previous benchmark. First, the average width grows with the model size. This is because the heating mode requires finer time partitioning than the cooling modes. The cooling modes increase with the number of rods, and so does the average width over all time partitions. Second, while with the filtered oscillator the difficulty laid at interpolation, for the rod reactor interpolation is rather easy as well as finding counterexamples. Most of the time is spent in the verification phase, where all fixpoint checks must be concluded, without being interrupted by a counterexample. This shows the advantage of our lazy approach, which first processes the counterexamples and finally proves the fixpoint.

Our method outperforms Ariadne on all benchmarks. On the other hand, tools like Flow* and SpaceEx can be dramatically faster [37]. For instance, they analyze `filtosc_8th_ord` in resp. 9.1s and 0.36s (time horizon of 4 and jump depth of 10). This is hardly surprising, as our method has primarily been designed to comply with soundness and time-unboundedness, and pays the price for that.

## 6.7 Related Work

There is a rich literature on CEGAR approaches for hybrid automata, either abstracting to a purely discrete system [10; 43; 98; 105; 107] or to a hybrid automaton with simpler dynamics [73; 101]. Both categories exploit the principle that the verification step is easier to carry out in the abstract domain. The abstraction entails a considerable loss of precision that can only be counteracted by increasing the number of abstract states. This leads to a state explosion that severely limits the applicability of such approaches. In contrast, our approach allows us to increase the precision by adding template directions, which does not increase the number of abstract states. The only case where we incur additional abstract states is when partitioning the time domain. This is a direct consequence of the nonconvexity of flowpipes of affine systems, and therefore seems to be unavoidable when using convex sets in abstractions. In [91], the abstraction consists of removing selected ODE entirely. This reduces the complexity, but does not achieve any fine-tuning between accuracy and complexity. Template reachability has been shown to be very effective in both scaling up reachability tasks to more efficient successor computations [104; 102; 57] and achieving termination even over unbounded time horizons [48]. The drawback of templates is the lack of accuracy, which may lead to an approximation error that accumulates excessively. Efforts to dynamically refine templates have so far not scaled well for affine dynamics [54]. A single-step refinement was proposed in [16], but as was illustrated in [23], the refinement needs to be inductive in order to exclude counterexamples in a CEGAR scheme.

# 7 Conclusions

We have investigated automatic abstraction refinement techniques for the time-unbounded reachability analysis of hybrid systems using template polyhedra. First, we have introduced a theory for the refinement of template directions from unfeasible paths coming, e.g., from a CEGAR loop. We have identified that, for abstractions whose post operators are defined in terms of Minkowski sums, linear transformations, conical hulls, maps through interval matrices, and linear or compact intersections, a refinement always exists and consists of at most one direction for each step in the path. The theory applies naturally to the abstraction of convex hybrid automata and, hence, to the special case of linear hybrid automata. As for hybrid automata with piece-wise affine dynamics, template refinement applies to overapproximations like linear phase approximations, which reduces them to convex hybrid automata, and interval arithmetic–based flowpipe approximations, which reduces them to discrete systems whose updates are given by interval matrices. Second, we have shown that, to construct a linear phase approximation whose pieces are unbounded in time and space, but are bounded in the aperture between derivatives, it is sufficient to partition the state space into cones. With this in mind, we have introduced the conic abstractions and, furthermore, shown that, for purely continuous diagonalizable systems, computing the abstraction terminates. Third, we have introduced the space-time interpolants, which couple the refinement of template directions with the refinement of the time partitioning, for flowpipe approximations based on interval arithmetic. In particular, we have developed a best-effort technique for computing space-time interpolants with a as-small-as-possible number of directions and time intervals that are precise enough to eliminate a counterexample.

Template refinement, conic abstractions, and space-time interpolants allow, in different ways, to refine the abstraction incrementally and automatically, so as to find an abstraction that is as coarse as possible, but precise enough to avoid spurious counterexamples. It turns out from our experiments that, unlike abstraction with fixed template and fixed time and space partitioning, reachability analysis, particularly over unbounded time, benefits from coarseness. Intuitively, the wider the abstraction, first, the easier (in our case) it is to compute and, second, the most likely (empirically) it finds a fixpoint. For convex hybrid automata, we demonstrated that searching for the smallest number of directions is orders of magnitude faster than using (and refining, if needed) a fixed template. In particular, refining the abstraction lazily enabled the practical time-unbounded reachability of quadratic hybrid automata and resulted in superior efficiency for linear hybrid automata. With the conic abstractions, we demonstrated that an appropriate conic partitioning guarantees coverage of the reach set for unbounded time, unlike typical flowpipe approximation methods, without sacrificing on precision. Finally, with the space-time interpolants, we demonstrated that using minimal templates and time partitioning allows flowpipe approximation methods, which were used before in a time-bounded fashion, to outperform the state-of-the-art tools for sound and automatic reachability over unbounded time.

Convex hybrid automata and hybrid automata with linear ODE can model various systems. Convex hybrid automata can model, e.g., timed systems whose drift between clocks is bounded within some Euclidean distance or approximate, e.g., stochastic systems whose disturbance is drawn from a (truncated) normal distribution; linear ODE can model various physical phenomena such as, e.g., heat transfer, motion. Proving the safety of these models can show that, e.g., a certain quantity never exceeds a critical threshold or that, e.g., a digital controller never enters an error state; our time-unbounded reachability analysis methods always provide a formal safety guarantee. Formal safety proofs, if practical, may constitute a fundamental step in the model-based design of an embedded hybrid systems, where systems are first modeled and, only after a successful analysis, deployed. Formal guarantees could be particularly important for systems that are safety critical, that is where a system's failure may endanger human lives.

We introduced methods for the refinement of template polyhedra for the reachability analysis of hybrid automata, posing the basis for future directions in terms of expressivity of the method, efficiency of the procedures, and applicability to other domains. For instance, as for expressivity, our method needs to be extended to piece-wise affine systems with non-deterministic inputs, which is challenging because it might involve larger post operators [53]; as for efficiency, our procedures recompute the abstraction after every refinement, while may be further optimized by constructing the abstraction incrementally [70]; as for applicability, our technique not only applies to hybrid automata, but, generally, to any similar abstract interpretation problem like, e.g., the verification of neural networks.

# Bibliography

[1] GLPK (GNU linear programming kit).

[2] A. Albarghouthi and K. L. McMillan. Beautiful interpolants. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 313–329, 2013.

[3] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Math. Program.*, 95(1):3–51, 2003.

[4] 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.

[5] M. Althoff, C. L. Guernic, and B. H. Krogh. Reachable set computation for uncertain time-varying linear systems. In *HSCC*, pages 93–102. ACM, 2011.

[6] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. 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, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*, pages 209–229. Springer, 1992.

[8] R. Alur, T. Dang, and F. Ivančić. Reachability analysis of hybrid systems via predicate abstraction. In *Hybrid Systems: Computation and Control (HSCC)*, pages 35–48. 2002.

[9] R. Alur, T. Dang, and F. Ivančić. Progress on reachability analysis of hybrid systems using predicate abstraction. In *HSCC*, pages 4–19. Springer, 2003.

[10] R. Alur, T. Dang, and F. Ivančić. Counterexample-guided predicate abstraction of hybrid systems. *Theoretical Computer Science*, 354(2):250–271, 2006.

[11] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[12] R. Alur and T. Henzinger. Modularity for timed and hybrid systems. *CONCUR'97: Concurrency Theory*, pages 74–88, 1997.

[13] R. Alur, T. A. Henzinger, and P. Ho. Automatic symbolic verification of embedded systems. In *RTSS*. IEEE Computer Society, 1993.

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

[15] E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *Computer Aided Verification*, pages 746–770. Springer, 2002.

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

[17] 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.

[18] C. Barrett and C. Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.

[19] G. Batt, C. Belta, and R. Weiss. Temporal logic analysis of gene networks under parameter uncertainty. *Transactions on Automatic Control*, 53(Special Issue):215–229, 2008.

[20] L. Benvenuti, D. Bresolin, P. Collins, A. Ferrari, L. Geretti, and T. Villa. Assume-guarantee verification of nonlinear hybrid systems with Ariadne. *Int. J. Robust. Nonlinear Control*, 24(4):699–724, 2014.

[21] J. A. Bergstra and C. A. Middelburg. Process algebra for hybrid systems. *Theoretical Computer Science*, 335(2-3):215–280, 2005.

[22] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in computers*, 58:117–148, 2003.

[23] S. Bogomolov, G. Frehse, M. Giacobbe, and T. A. Henzinger. Counterexample-guided refinement of template polyhedra. In *TACAS (1)*, volume 10205 of *Lecture Notes in Computer Science*, pages 589–606, 2017.

[24] S. Bogomolov, M. Giacobbe, T. A. Henzinger, and H. Kong. Conic abstractions for hybrid systems. In *FORMATS*, volume 10419 of *Lecture Notes in Computer Science*, pages 116–132. Springer, 2017.

[25] S. Bogomolov, C. Herrera, and W. Steiner. Benchmark for verification of fault-tolerant clock synchronization algorithms. In *ARCH (2016).*

[26] O. Botchkarev and S. Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. *HSCC*, pages 73–88, 2000.

[27] O. Bournez, O. Maler, and A. Pnueli. Orthogonal polyhedra: Representation and computation. In *HSCC*, volume 1569 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 1999.

[28] S. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

[29] T. Brihaye, L. Doyen, G. Geeraerts, J. Ouaknine, J.-F. Raskin, and J. Worrell. On reachability for hybrid automata over bounded time. In *International Colloquium on Automata, Languages, and Programming*, pages 416–427. Springer, 2011.

[30] L. Bu, Y. Li, L. Wang, and X. Li. BACH : Bounded reachability checker for linear hybrid automata. In *FMCAD*, pages 1–4. IEEE, 2008.

[31] L. Bu, J. Zhao, and X. Li. Path-oriented reachability verification of a class of nonlinear hybrid automata using convex programming. In *VMCAI*, 2010.

[32] T. Chen, N. Yu, and T. Han. Continuous-time orbit problems are decidable in polynomial-time. *Inf. Process. Lett.*, 115(1):11–14, 2015.

[33] X. Chen and E. Ábrahám. Choice of directions for the approximation of reachable sets for hybrid systems. In *EUROCAST (1)*, volume 6927 of *Lecture Notes in Computer Science*, pages 535–542. Springer, 2011.

[34] X. Chen, E. Abraham, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 183–192. IEEE, 2012.

[35] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *RTSS'12*, pages 183–192, 2012.

[36] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *CAV*, 2013.

[37] X. Chen, S. Schupp, I. B. Makhlouf, E. Ábrahám, G. Frehse, and S. Kowalewski. A benchmark suite for hybrid systems reachability analysis. In *NASA Formal Methods Symposium*, pages 408–414. Springer, 2015.

[38] V. Chonev, J. Ouaknine, and J. Worrell. The orbit problem in higher dimensions. In *STOC*, pages 941–950. ACM, 2013.

[39] V. Chonev, J. Ouaknine, and J. Worrell. The polyhedron-hitting problem. In *SODA*, pages 940–956. SIAM, 2015.

[40] A. Chutinan and B. H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *HSCC*, volume 1569 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 1999.

[41] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. Hycomp: An smt-based model checker for hybrid systems. In *TACAS*, volume 9035 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 2015.

[42] A. Cimatti, S. Mover, and S. Tonetta. A quantifier-free SMT encoding of non-linear hybrid automata. In *FMCAD*, 2012.

[43] E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *International Journal of Foundations of Computer Science*, 14(04):583–604, 2003.

[44] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV*, 2000.

[45] P. Collins, D. Bresolin, L. Geretti, and T. Villa. Computing the evolution of hybrid systems using rigorous function calculus. *IFAC Proceedings Volumes*, 45(9):284–290, 2012.

[46] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, 1977.

[47] P. J. L. Cuijpers and M. A. Reniers. Hybrid process algebra. *The Journal of Logic and Algebraic Programming*, 62(2):191–245, 2005.

[48] T. Dang and T. M. Gawlitza. Template-based unbounded time verification of affine hybrid automata. In *Asian Symposium on Programming Languages and Systems*, pages 34–49. Springer, 2011.

[49] L. Doyen, T. A. Henzinger, and J.-F. Raskin. Automatic rectangular refinement of affine hybrid systems. In *Proceedings of the Third International Conference on Formal Modeling and Analysis of Timed Systems*, FORMATS'05, pages 144–161, Berlin, Heidelberg, 2005. Springer-Verlag.

[50] A. Eggers, M. Fränzle, and C. Herde. SAT modulo ODE: A direct SAT approach to hybrid systems. In *ATVA*, volume 5311 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 2008.

[51] A. Fehnker and F. Ivančić. Benchmarks for hybrid systems verification. In *International Workshop on Hybrid Systems: Computation and Control*, pages 326–341. Springer, 2004.

[52] G. Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *International workshop on hybrid systems: computation and control*, pages 258–273. Springer, 2005.

[53] G. Frehse. Reachability of hybrid systems in space-time. In *EMSOFT*, pages 41–50. IEEE, 2015.

[54] G. Frehse, S. Bogomolov, M. Greitschus, T. Strump, and A. Podelski. Eliminating spurious transitions in reachability with support functions. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 149–158. ACM, 2015.

[55] G. Frehse, M. Giacobbe, and T. A. Henzinger. Space-time interpolants. In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, pages 468–486. Springer, 2018.

[56] 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.

[57] 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 *CAV'11*, pages 379–395, 2011.

[58] A. Girard. Reachability of uncertain linear systems using zonotopes. *HSCC*, pages 291–305, 2005.

[59] C. L. Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In *CAV*, 2009.

[60] E. Hainry. Reachability in linear dynamical systems. In *CiE*, volume 5028 of *Lecture Notes in Computer Science*, pages 241–250. Springer, 2008.

[61] N. Halbwachs, Y.-E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *International Static Analysis Symposium, SAS'94*, Namur (Belgium), September 1994.

[62] J. Hanson. Rotations in three, four, and five dimensions. *arXiv preprint arXiv:1103.5263*, 2011.

[63] T. Henzinger. The theory of hybrid automata. In *Proc. IEEE Symp. Logic in Computer Science*, pages 278–292, 1996.

[64] 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.

[65] T. Henzinger and H. Wong-Toi. Linear phase-portrait approximations for nonlinear hybrid systems. *Hybrid Systems III*, pages 377–388, 1996.

[66] T. A. Henzinger and P. Ho. HYTECH: the cornell hybrid technology tool. In *Hybrid Systems*, volume 999 of *Lecture Notes in Computer Science*, pages 265–293. Springer, 1994.

[67] T. A. Henzinger and P.-H. Ho. A note on abstract interpretation strategies for hybrid automata. In *International Hybrid Systems Workshop*, 1994.

[68] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *International Conference on Computer Aided Verification*, pages 460–463. Springer, 1997.

[69] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE transactions on automatic control*, 43(4):540–554, 1998.

[70] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *POPL*, 2002.

[71] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *J. Comput. Syst. Sci.*, 57(1):94–124, 1998.

[72] M. W. Hirsch, S. Smale, and R. L. Devaney. *Differential equations, dynamical systems, and an introduction to chaos.* Academic press, 2012.

[73] S. K. Jha, B. H. Krogh, J. E. Weimer, and E. M. Clarke. Reachability for linear hybrid automata using iterative relaxation abstraction. In *HSCC*, 2007.

[74] Y. Jiang, H. Song, R. Wang, M. Gu, J. Sun, and L. Sha. Data-centered runtime verification of wireless medical cyber-physical system. *IEEE Transactions on Industrial Informatics*, 2016.

[75] Y. Jiang, H. Zhang, Z. Li, Y. Deng, X. Song, M. Gu, and J. Sun. Design and optimization of multiclocked embedded systems using formal techniques. *IEEE transactions on industrial electronics*, 62(2):1270–1278, 2015.

[76] F. Johansson. Arb: efficient arbitrary-precision midpoint-radius interval arithmetic. *IEEE Transactions on Computers*, 66:1281–1292, 2017.

[77] R. Kannan and R. J. Lipton. Polynomial-time algorithm for the orbit problem. *J. ACM*, 33(4):808–821, 1986.

[78] M. Kloetzer and C. Belta. Reachability analysis of multi-affine systems. In *HSCC*, pages 348–362. Springer, 2006.

[79] H. Kong, E. Bartocci, S. Bogomolov, R. Grosu, T. A. Henzinger, Y. Jiang, and C. Schilling. Discrete abstraction of multiaffine systems. In *International Workshop on Hybrid Systems Biology*, pages 128–144. Springer, 2016.

[80] H. Kong, F. He, X. Song, W. N. Hung, and M. Gu. Exponential-condition-based barrier certificate generation for safety verification of hybrid systems. In *CAV*, pages 242–257. Springer, 2013.

[81] S. Kong, S. Gao, W. Chen, and E. M. Clarke. dreach: $\delta$-reachability analysis for hybrid systems. In *TACAS*, volume 9035 of *Lecture Notes in Computer Science*, pages 200–205. Springer, 2015.

[82] A. B. Kurzhanski and P. Varaiya. *Dynamics and Control of Trajectory Tubes*. Springer, 2014.

[83] G. Lafferriere, G. J. Pappas, and S. Sastry. O-minimal hybrid systems. *MCSS*, 13(1):1–21, 2000.

[84] G. Lafferriere, G. J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In *HSCC*, volume 1569 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 1999.

[85] L. Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems (TOCS)*, 5(1):1–11, 1987.

[86] C. Le Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In *International Conference on Computer Aided Verification*, pages 540–554. Springer, 2009.

[87] N. Lynch, R. Segala, and F. Vaandrager. Hybrid i/o automata. *Information and computation*, 185(1):105–157, 2003.

[88] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)*, pages 447–484. Springer, 1991.

[89] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review*, 45(1):3–49, 2003.

[90] MOSEK ApS. *The MOSEK C optimizer API manual. Version 7.1 (Revision 53).*, 2015.

[91] J. Nellen, E. Ábrahám, and B. Wolters. A cegar tool for the reachability analysis of plc-controlled plants using hybrid automata. In *Formalisms for Reuse and Systems Integration*, pages 55–78. Springer, 2015.

[92] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In *Hybrid Systems*, pages 149–178. Springer, 1992.

[93] A. Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *Journal of Logic and Computation*, 20(1):309–352, 2008.

[94] A. Platzer. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning*, 41(2):143–189, 2008.

[95] A. Platzer and J. Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *IJCAR*, volume 5195 of *Lecture Notes in Computer Science*, pages 171–178. Springer, 2008.

[96] M. V. Ramana. An exact duality theory for semidefinite programming and its complexity implications. *Math. Program.*, 77, 1997.

[97] S. Ratschan. Safety verification of non-linear hybrid systems is quasi-semidecidable. In *International Conference on Theory and Applications of Models of Computation*, pages 397–408. Springer, 2010.

[98] S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Transactions on Embedded Computing Systems (TECS)*, 6(1):8, 2007.

[99] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, 1970.

[100] J. Rohn. Systems of linear interval equations. *Linear algebra and its applications*, 126:39–78, 1989.

[101] N. Roohi, P. Prabhakar, and M. Viswanathan. Hybridization based cegar for hybrid automata with affine dynamics. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 752–769. Springer, 2016.

[102] S. Sankaranarayanan, T. Dang, and F. Ivancic. Symbolic model checking of hybrid systems using template polyhedra. In *TACAS*, 2008.

[103] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *VMCAI*, 2005.

[104] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 25–41. Springer, 2005.

[105] M. Segelken. Abstraction and counterexample-guided construction of $\omega$-automata for model checking of step-discrete linear hybrid models. In *International Conference on Computer Aided Verification*, pages 433–448. Springer, 2007.

[106] A. Sogokon, K. Ghorbal, P. B. Jackson, and A. Platzer. A method for invariant generation for polynomial continuous systems. In *TACAS*, pages 268–288. Springer, 2016.

[107] M. Sorea. Lazy approximation for dense real-time systems. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 363–378. Springer, 2004.

[108] L. Tavernini. Differential automata and their discrete simulators. *Nonlinear Analysis: Theory, Methods & Applications*, 11(6):665–683, 1987.

[109] A. Tiwari. Abstractions for hybrid systems. *Formal Methods in System Design*, 32(1):57–83, 2008.

[110] A. Tiwari and G. Khanna. Series of abstractions for hybrid automata. In *HSCC*, pages 465–478. Springer, 2002.

[111] F. Vaandrager. Hybrid systems. *Images of SMC Research*, pages 305–316, 1996.