Theory of Hybrid Systems Informatik 2

RWTH AACHEN UNIVERSITY

Diese Arbeit wurde vorgelegt am LuFG Theorie hybrider Systeme

MASTERARBEIT

REACHABILITY ANALYSIS FOR HYBRID AUTOMATA WITH URGENT JUMPS

Maria Kazantzi

Communicated by
Prof. Dr. Erika Ábrahám

Prüfer:
Prof. Dr. Erika Ábrahám
Apl. Prof. Dr. Thomas Noll

Zusätzlicher Berater:
József Kovacs

Aachen, September 5, 2024

Abstract

Hybrid automata are a formalism for modeling systems with discrete and continuous behavior. To check the safety of a hybrid system with non-linear behavior, we calculate its set of reachable states. This thesis considers hybrid automata with urgent transitions, which require immediate action when the transition is enabled. More precisely, we extend the HyPro library to be able to analyze the reachability of urgent automata. Analyzing hybrid systems with non-linear behavior is not straightforward. One possibility is through over-approximative flowpipe construction based on reachability analysis. Over-approximation ensures that all possible states the system can reach are included in the flowpipe and provide a robust safety check. Over-approximative calculations are useful for safety verification. Another method that is extremely useful for analyzing hybrid systems is under-approximative flowpipe computations. The main goal of this thesis is to develop and present a method for computing the flowpipes of a hybrid automaton with urgency, by over-approximating the set minus operation, which is essential for urgent jumps. A critical aspect of this process is the computation of the convex set minus of two polytopes. The set minus operator is essential for the flowpipe construction method. The algorithm proposed in this thesis is designed to handle the complexity of urgent automata and ensures precise over-approximation of reachable states. We validate our approach through a series of experiments and present results that demonstrate the effectiveness and functionality of our method. This thesis contributes to the field of hybrid automata by providing methods for analyzing urgent automata and ensuring the safety of these systems.

Contents

1 Introduction **9**

2 Related Work **11**

3 Preliminaries **13**
3.1 Hybrid Systems . . . 13
3.2 State Set Representations . . . 15
3.3 Reachability Analysis . . . 18

4 HyPro Library **23**

5 Computation of Set Minus Operator **25**
5.1 Over-approximation . . . 25
5.1.1 Motivation . . . 25
5.1.2 Definitions and Lemmas . . . 29
5.1.3 Proof of Optimality . . . 37
5.1.4 Algorithm . . . 38
5.1.5 Runtime . . . 42
5.2 Under-approximation . . . 46
5.2.1 Motivation . . . 46
5.2.2 Example . . . 46
5.2.3 Algorithm . . . 47

6 Results **51**
6.1 Over-approximation Results . . . 51
6.1.1 First Example . . . 51
6.1.2 Second Example . . . 52
6.1.3 Third Example . . . 53
6.2 Under-approximation Results . . . 54
6.3 Runtime Comparison Results . . . 54
6.3.1 First Model . . . 55
6.3.2 Second Model . . . 56

7 Conclusion **59**
7.1 Summary . . . 59
7.2 Future work . . . 60

Bibliography **63**

Chapter 1

Introduction

Hybrid automata are a powerful tool used for modeling systems that have both discrete and continuous behavior. They accurately represent many real-world systems which such behavior. One example of such a system is a bouncing ball, where the ball's motion is continuous but the bounce with the ground is a discrete event. Another example is a thermostat, which continuously monitors the temperature but switches the heating or cooling system. This switch is a discrete change in the system's behavior. Hybrid automata are also used in more complex systems such as automotive control systems, aerospace navigation systems, medical equipment, robotics, and many others [3]. Such complex systems, require safety and good performance. It is essential to ensure, that the system does not reach bad or unsafe states, which could lead to harmful outcomes for people or the environment.

In the context of hybrid automata, ensuring system safety involves analyzing the reachable state sets of an automaton over time. One of the techniques to do so, is through flowpipe construction. This technique uses geometric state set representation to represent all possible states the system can reach as time evolves. Flowpipes are crucial for visualizing and verifying the state space of a system, allowing one to determine whether a system might enter an unsafe set of states. The key challenge is to ensure that the constructed flowpipe accurately includes all potential behaviors of the system. Safety analysis typically focuses on over-approximating the reachable states to guarantee that all possible behaviors are captured. Over-approximation ensures that if the flowpipe does not intersect with an unsafe set of states, then the actual system is safe. Conversely, under-approximation, which captures only a subset of possible behaviors, can be useful for identifying specific safe behaviors. However, under-approximation is not suitable for safety verification, as it may miss unsafe state sets because it does not include all possible behaviors of the system.

In many real-world applications, certain transitions must occur immediately when specific conditions are met. An example for that, is a vehicle that has to brake in certain positions. When the vehicle reaches these positions, it must brake for a second and then continues moving. These types of transitions are known as urgent. Traditional hybrid automata models do not consider such urgency, potentially leading to inaccuracies in the safety analysis. In this thesis, we introduce the concept of urgent hybrid automata, which extends traditional hybrid automata to include urgent transitions. In urgent hybrid automata, urgent transitions are prioritized over other transitions, ensuring that they are executed immediately when their conditions

are met. This allows for more accurate modeling of systems that require immediate responses to certain events. For example, in an emergency braking system of a car, a transition from normal driving to emergency braking must occur immediately upon detecting an obstacle at a close distance. Traditional models might delay this transition, failing to capture the urgency required for accurate safety analysis.
To analyze the reachability analysis of urgent hybrid automata, we need to modify the flowpipe construction process to handle immediate transitions effectively. A critical mathematical operation in this analysis is the convex set minus operation of two convex sets.
The main goal of this thesis is to create an algorithm to compute the convex set minus of two polytopes. In our approach, we focus on the set minus operator using over-approximation. However, we also present a method for the set minus operator using under-approximation, to identify specific safe trajectories.
The thesis begins with a chapter of related work 2 that provides an overview of existing methodologies for analyzing hybrid automata, with a focus on urgent transitions and reachability analysis. We then introduce in Chapter 3 the necessary mathematical background and preliminaries. In Chapter 4, we mention some background context about the HyPro library that is used and extended in this thesis. In Chapter 5, we introduce the concept of a convex set minus operator of two convex sets. Afterward, we present three different example scenarios and how the result of the convex set minus operator is in each case. Before introducing the pseudocode, we provide additional definitions and lemmas to prove the correctness of the algorithm. Then we present in detail the implemented code and analyze the runtime complexity of the algorithm. In Section 5.2, a method for under-approximating the flowpipes is presented. We discuss the motivation behind the under-approximation and continue with an example. At the end of that section, we show the algorithm that is implemented in the library. Lastly, in Chapter 6, the experimental results are presented. These results have been created and used to show the correctness and efficiency of the proposed algorithm. Three different cases of the over-approximation algorithm are illustrated. We show the difference on the flowpipes when the jumps are urgent with the usage of the implemented algorithm. We continue with a section for the results of the under-approximation algorithm. The last part of this chapter is about the comparison of runtime between the introduced algorithm with another algorithm that is going to be presented in Chapter 2 and is already implemented in the HyPro library. For the comparison, we use two different models and we present the runtime results in a table and plots. Lastly, in Chapter 7, we summarize this thesis and discuss potential future work.

Chapter 2

Related Work

In the field of hybrid automata analysis, various methodologies have been developed to deal with the challenges of urgency and reachability. This section reviews relevant literature, focusing on methodologies that align with the goals of this thesis. Specifically, developing polytope set operations is important in proving the accuracy and efficiency of hybrid automata verification. With polytopes, the reachable states of a hybrid automaton at a given time are represented. In order to incorporate urgency into the reachability analysis, the set minus operation is essential. In [9], the authors concentrate their efforts on advancing the field of formal verification, specifically with a focus on Linear Hybrid Automata (LHA). In their work, they address the limited capacity to handle convex invariants and urgency conditions defined by single constraints. These limitations become apparent when LHAs are employed to model urgent transitions in deterministic languages like Matlab-Simuling [2], Modelica [10], and Ptolemy [14], where all transitions are urgent. The paper [9] proposes an algorithm designed to overcome these challenges, enabling successor computation with non-convex invariants and closed, linear urgency conditions. In [7], the author presents an alternative strategy for addressing non-convex invariants. The algorithm requires transforming the automaton by splitting locations to model non-convex invariants. In paper [9], the approach aims to address these limitations more directly and avoid building a new automaton. Hybrid systems with urgency are also explored in [11]. In this paper, the idea is to create an automaton equivalent to the original but without the urgent transitions. To achieve this, each location with an outgoing urgent transition is replaced by multiple new locations. The invariants of these new locations are extended by the inverted halfspaces of the guard. This approach ensures that time can only progress while the interior of the guard is not entered.

In [5], the focus lies on extending the flowpipe construction to accommodate the urgency to ensure safety verification for urgent hybrid automata. This approach aligns closely with the objectives of this thesis, which also aims to develop a set minus operator for analyzing urgent hybrid automata. While this thesis uses over-approximation and under-approximation to compute the set minus operator, the work in [5] focuses on computing the exact set minus of two polytopes. The setMinus-Polytopes algorithm proposed in [8] and used in [5] is the result of this research. We refer to the exact set minus operation as setMinus-Polytopes in this thesis. This method processes two H-Polytopes as input, which represent the state sets of hybrid automata, and computes their set difference. The core idea behind the setMinus-Polytopes algorithm is

to iteratively explore the feasible region of a resulting H-Polytope by systematically reversing one constraint of the guard polytope at each iteration. After each constraint reversal, the function uses linear programming techniques to evaluate the feasibility of the solution space. The feasible solutions in each iteration are stored in a vector that contains the resulting polytopes. An important aspect of this algorithm, which can also affect the size of the result, is the order in which the constraints are inverted. A significant disadvantage of this approach is that after a single set minus step, the resulting vector includes multiple polytopes. In subsequent set minus operations, this can lead to an exponential increase in the number of polytopes, potentially resulting in reachability tree explosion, regarding the number of branches.
Rather than computing the exact set minus of two polytopes, this thesis aims to compute a single polytope, which is an over-approximation or under-approximation of two polytopes using the set minus operation. This mitigates the risk of reachability tree explosion and simplifies the computation process. However, the setMinus-Polytopes algorithm is more accurate than the over-approximation and under-approximation algorithms developed in this thesis.

Chapter 3

Preliminaries

This chapter begins with the fundamentals and provides the necessary definitions to establish a solid foundation for the rest of the thesis. We will start by introducing the basic concepts, including mathematical definitions and theoretical frameworks. Additionally, we will use examples to clarify the concepts, ensuring a better understanding of the key definitions.

3.1 Hybrid Systems

We begin by introducing hybrid automata, which are used to model hybrid systems. Hybrid automata describe the states of a hybrid system using variables and locations. These systems form the foundation of this thesis.

Definition 3.1.1 (Hybrid systems)**.** Hybrid systems are dynamic systems that exhibit both continuous and discrete behavior. Dynamical behavior describes a continuous change of state over time, whereas discrete behavior describes instantaneous state changes.

Hybrid systems illustrate various examples such as a bouncing ball, a thermostat, a car, a computer, or a robot. For instance, consider the bouncing ball. As the ball travels through the air, the position and velocity of the ball change continuously due to gravity. However, when the ball hits the ground, it undergoes a discrete change in velocity due to its impact on the ground.
In order to describe and model hybrid systems, we use hybrid automata. We define hybrid automata as follows:

Definition 3.1.2 (Hybrid automaton)**.** Hybrid automaton is a formal model for hybrid systems which is described as a tuple $H = (Loc, Var, Lab, Edge, Act, Inv, L_0)$, where:

- Loc is a finite set of locations,

- Var is a finite set of variables,
- Lab is a finite set of labels,
- $Edge \subseteq Loc \times Lab \times 2^{(V^2)} \times Loc$ is a finite set of edges,
- Act is a function that maps a set of activities $f : \mathbb{R}^+ \to V$ to each location; the activity sets are time-invariant. Time invariant means that $f \in Act(l)$ implies $(f + t) \in Act(l)$, where $(f + t)(t') = f(t + t')$ for all $t' \in \mathbb{R}^+$,
- Inv is a function that maps an invariant $Inv(I) \subseteq V$ to each location $I \in Loc$,
- $L_0 \subseteq Loc$ is the set of the initial states of the automaton

with valuations $v : Var \to \mathbb{R}$, V is the set of valuations. Valuation refers to the assignment of values to variables within a system.

Figure 3.1 models the bouncing ball example as a hybrid automaton. The variables used to describe the hybrid automaton are $Var = \{pos,vel\}$. The variable pos describes the position of the ball and vel the velocity. In the beginning, the ball is above the ground in position pos and it starts falling with a certain velocity. The derivatives $\dot{pos}$ and $\dot{vel}$ describe how the position and velocity change over time. While the ball is falling, the velocity decreases with the gravity constant g. When the ball hits the ground, it bounces back up with the same velocity as before, but in the opposite direction. The velocity is scaled down by a constant c, which can be selected from the interval [0,1]. This happens in order to model the loss of energy due to the inelastic nature of real-world collisions. When the ball bounces, it does not retail all its initial energy [1].

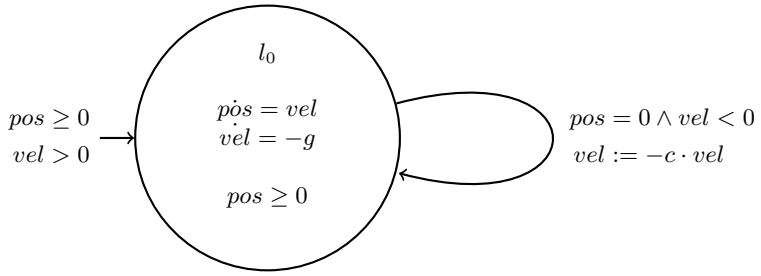


Figure 3.1: A hybrid automaton for the bouncing ball.

This thesis focuses on automata with urgent transitions. Urgency means that the transitions must be executed immediately upon the occurrence of a condition. With this consideration, we broaden the definition of hybrid automata to include urgent transitions. We define urgent automata as follows:

Definition 3.1.3 (Urgent automaton)**.** An urgent hybrid automaton is a tuple $H = (Loc, Var, Lab, Edge, Act, Inv, L_0, Urg)$ such that:

- $Urg \subseteq Edge$ is a set of urgent transitions.
- $(Loc, Var, Lab, Edge, Act, Inv, L_0)$ is a hybrid automaton.

3.2 State Set Representations

Building upon our exploration of hybrid systems and automata in the previous section, we now shift our focus to the geometric representation of the states within these systems. The state-set representation forms a critical aspect of modeling and analyzing hybrid systems. One way of modeling a state set is using polytopes.

Definition 3.2.1 (Polyhedron)**.** A polyhedron is a geometric space formed by faces, edges, and vertices. A polyhedron in $\mathbb{R}^d$ is the solution set to a finite number of linear inequalities, for $d \in \mathbb{N}_{\geq 0}$.

A polyhedron is considered bounded if it is contained within some finite region of space, meaning it does not extend infinitely in any direction. When a polyhedron is bounded, it is called a polytope.

We define $\mathbb{U}_P^d$ as the set of all convex polyhedra in $\mathbb{R}^d$.

Due to polyhedra's structural characteristics, they demonstrate the properties of convexity. Any two points inside a polyhedron can be connected by a straight line, where each point on that line lies within the polyhedron.

Definition 3.2.2 (Convex sets)**.** A set $S \subseteq \mathbb{R}^d$ for $d \in \mathbb{N}_{\geq 0}$ is convex if for every pair of points $x, y \in S$ and $\lambda \in [0, 1]$, it holds:

$$\lambda \cdot x + (1 - \lambda) \cdot y \in S$$

Definition 3.2.3 (Convex hull)**.** The convex hull of a set of points $S \subseteq \mathbb{R}^n$ is the smallest convex set that contains S.

To be able to represent polyhedra effectively and utilize them in the context of hybrid systems, we introduce two primary representations: the H-Representation and the V-Representation. These representations enable algorithmic manipulation of polyhedra

by facilitating operations such as intersection, union, and projection. Also, they often lead to more efficient computations and optimized performance in geometric set operations.

Definition 3.2.4 (Halfspace Representation)**.** The H-Representation describes a polytope as the intersection of halfspaces defined by linear inequalities. Let P be a polytope. The H-Representation of P is given by:

$$P = \{x \in \mathbb{R}^d | Ax \leq b\}$$

where $A \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^m$, for some $m \in \mathbb{N}$ and $d \in \mathbb{N}$.
We call (A,b) an H-Polyhedron, or if the polyhedron is bounded, an H-polytope.
For a set of points $X \subseteq \mathbb{R}^d$, we define H-P(X) as a convex H-Polytope constructed by transforming X into a set of linear inequalities.
We define $\mathbb{U}^d_{P_H}$ as the set of all H-Polyhedra in $\mathbb{R}^d$.

Definition 3.2.5 (Vertex Representation)**.** Let $X \subseteq \mathbb{R}^d$ be a set of points for some dimension $d \in \mathbb{N}$.
The V-Representation is used to describe a polytope as a convex hull of a set of points.
We define the following notations:

- V-P(X) as a convex V-Polytope constructed from a set of points X.
- $\mathbb{U}^d_{P_V}$ as the set of all convex V-Polytopes in $\mathbb{R}^d$.

The runtime for converting between the two representations depends on several factors, including the dimensionality of the polytopes and the number of vertices, or constraints. In general, the process can be computationally intensive, especially for high-dimensional polytopes. It is mentioned in [1], that the translations between H- and the V-Representations of polytopes can be exponential in the d dimensional state space.
To visualize the different representation types, Figure 3.2 provides an example of a bounded polytope. The solution space is highlighted in blue.
The H-Representation of the state space is:

$$P = \{x_1 + x_2 \geq 3, 2x_1 - x_2 \leq 5, -x_1 + 2x_2 \leq 3\}$$

and the V-Representation is:

$$P = \{\begin{pmatrix}1\\2\end{pmatrix}, \begin{pmatrix}\frac{8}{3}\\\frac{1}{3}\end{pmatrix}, \begin{pmatrix}\frac{13}{3}\\\frac{11}{3}\end{pmatrix}\}$$

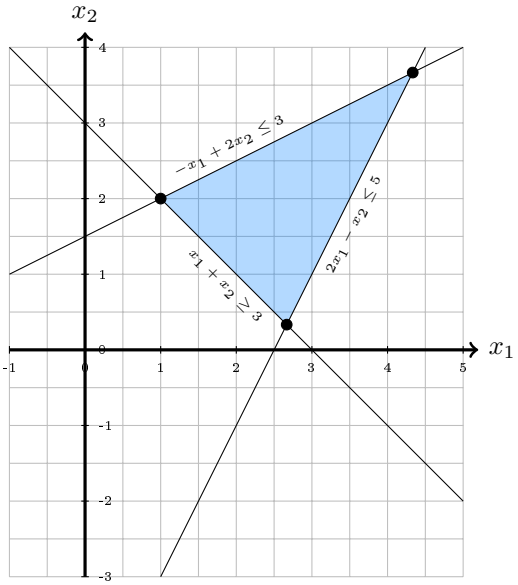


Figure 3.2: Example of a polytope.

State set representations such as H- and V-Representations allow us to represent the valuation set of the variables as polytopes. In order to analyze and describe polytopes in more detail, we introduce the concept of faces. Faces are the building blocks of polytopes and provide a way to describe their structure.

Definition 3.2.6 (Faces)**.** A face of a convex polytope $P \subseteq \mathbb{R}^d$ for some $d \in \mathbb{N}$ is defined as the intersection of P with any halfspace such that the boundary of this halfspace does not intersect the interior of P. For this thesis, we use the following terms [17]:

- Vertices (0-dimensional faces): The 0-dimensional faces of P include all "corner" points of P.
- Edges (1-dimensional faces): The 1-dimensional faces of P consist of all edges connecting the 0-dimensional faces of P.

In this thesis, we assume that the term 1-dimensional faces(P) is bidirectional, meaning for all u,$v \in$ 0-dimensional faces(P):

$$(u,v) \in \text{1-dimensional faces}(P) \text{ and } (v,u) \in \text{1-dimensional faces}(P)$$

In Figure 3.2.6, we visualize an example of a polytope with 5 vertices and 8 edges.

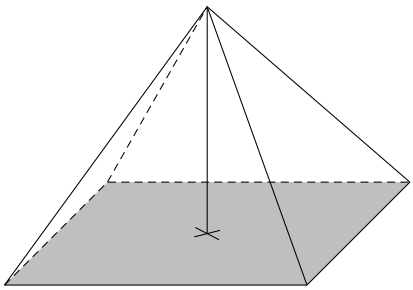


Figure 3.3: Example of polytope faces (vertices, edges) in $\mathbb{R}^3$.

Given a polytope, which represents the state set on a location, one interesting question arises. Namely, whether a single instance (denoted as a point $p \in \mathbb{R}^d$) is a member of the polytope or not. For that, we use a membership function [1], which is defined as follows:

Definition 3.2.7 (Member)**.** Let $P \subseteq \mathbb{R}^d$ be a polytope with $d \in \mathbb{N}_{\geq 0}$ and $p \in \mathbb{R}^d$ a point. We call p a member of P iff:

P is an H-Polytope (A,b):

$$p \text{ member } P \Leftrightarrow A \cdot p \leq b$$

P is a V-Polytope with $P = \{v_1,...,v_k\}$:

$$p \text{ member } P \Leftrightarrow \exists \lambda_1,..., \lambda_k \in [0,1] \subseteq \mathbb{R} : \sum_{i=1}^{k} \lambda_i = 1 \wedge p = \sum_{i=1}^{k} \lambda_i \cdot v_i$$

3.3 Reachability Analysis

Reachability analysis shows if the hybrid system is safe or not. This analysis provides information about all the states a system can reach within a finite time. An important aspect of reachability analysis involves constructing a flowpipe by piecing together flowpipe segments. Flowpipe segments are used in order to avoid a strong over-approximation of the trajectory of the system. Another reason for using flowpipe segments is that the systems are often non-linear and complex. By using flowpipe segments, the over-approximation of the trajectory is more accurate. Instead of analyzing the entire flowpipe as a single entity, we divide it into smaller segments. Each of these segments captures the system's behavior over a time interval. This results in a more accurate over-approximation of the flowpipe with much less error.

Definition 3.3.1 (Flowpipe)**.** Let T be a time point, the flowpipe is the over-approximation of the set of states that the system can reach in a fixed time interval $[0,T]$ from a set of initial states.

Reachability analysis of hybrid automata refers to the ability to determine the set of states a system can reach from a set of initial states to a finite time point. The reachability analysis in hybrid systems aims to discover the set of possible states the system can enter over time, taking into account both continuous and discrete behaviors.
In Figure 3.4, an example is presented to illustrate the reachability analysis of the hybrid automaton from Figure 3.1. The plot illustrates the continuous states the model reaches as time evolves. The boxes in the figure indicate the various states that the model is in at a specific time. In addition, the red box signifies the bad states, which are initially given. The x-axis describes the velocity of the ball, and the y-axis shows the position of the ball. The reachability analysis helps to identify whether the system can enter any of these bad states given its initial conditions.

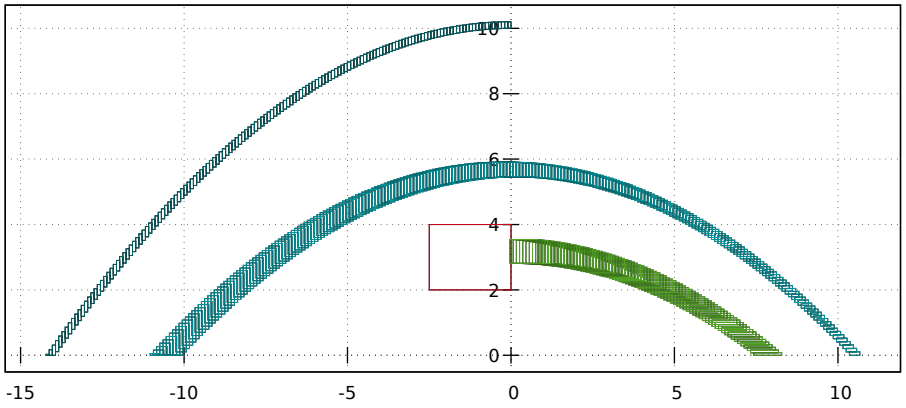


Figure 3.4: Reachable states of the bouncing ball example.

Algorithm 1 shows the flowpipe-construction-based reachability analysis, which is presented in [1]. The idea of Algorithm 1 is to start with (an) initial state set(s) and iteratively compute successors of these states by alternating between flow and jump successor computations. R contains the reachable state sets and R^{new} the state sets to be processed. The algorithm repeatedly selects unprocessed state sets from R^{new} and computes the flowpipe for each set using the computeFlowPipe method. This returns a set R' covering all states reachable within a given time horizon. If a jump depth is not reached, the jump successors of R' using the computeJumpSucc method

are computed. The algorithm then adds the new states to R and continues until a fixed point is reached or the jump depth is reached. The algorithm returns the set of reachable states R.

Algorithm 1 Forward reachability analysis

Input: Initial set $Init$
Output: Set R of reachable states

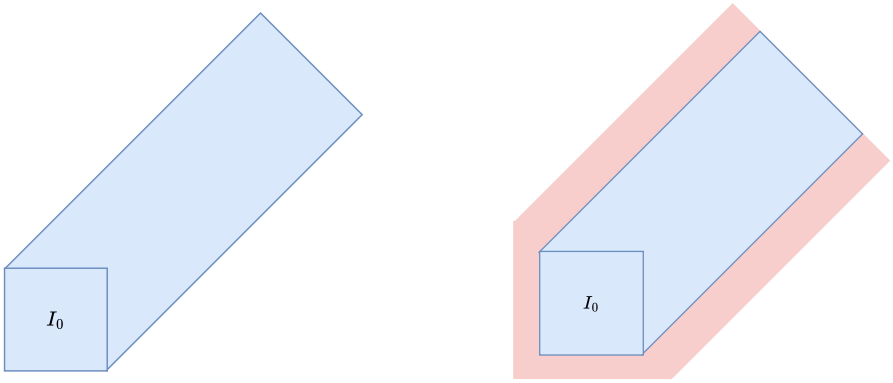
1: $R := Init$
2: $R^{new} := Init$
3: **while** $R^{new} \neq \emptyset$ **do**
4: Let $stateSet \in R^{new}$
5: $R^{new} := R^{new} \setminus \{stateSet\}$
6: $R' := computeFlowPipe(stateset)$
7: **if** $!jumpDepthReached()$ **then**
8: $R^{new} := R^{new} \cup computeJumpSucc(R')$
9: **end if**
10: $R := R \cup R^{new}$
11: **end while**
12: **return** R

An over-approximation is a method to over-approximate the reachable sets that a system can reach over time. The over-approximation provides a superset of the reachable states of a system. By over-approximating, is often more conservative than the exact representation, meaning that it may include states that are not reachable by the system. However, it is useful as it ensures that the behavior of the system is safe and avoids potential errors in the analysis.

Definition 3.3.2 (Over-Approximation)**.** Given a reachable set $S(t,I_0)$, where $t \in \mathbb{R}^+$ is the time and $I_0 \subseteq \mathbb{R}^d$ is the initial set of states for some $d \in \mathbb{N}$, the over-approximation set $\overline{S}(t,I_0)$ is defined as [16]:

$$S(t,I_0) \subseteq \overline{S}(t,I_0)$$

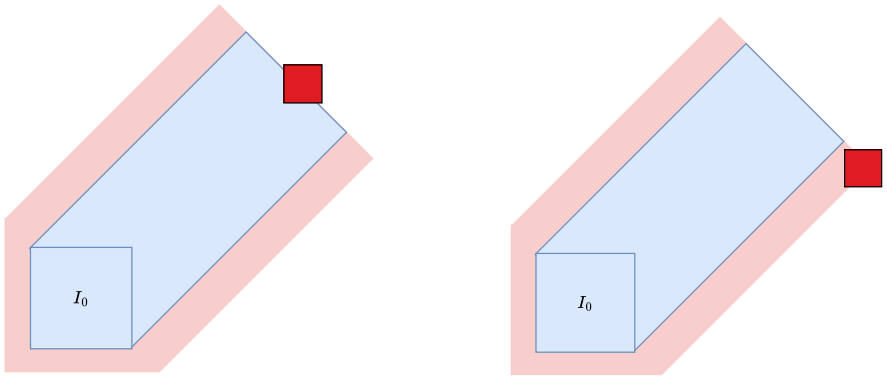
In Figure 3.5a the reachable states of a hybrid system from the initial state set I_0 in t time steps are shown. Figure 3.5b visualizes the difference to an over-approximation of that set, which is a superset of the reachable states. The over-approximation set $\overline{S}(t,I_0)$ is the red area together with the blue reachable set $S(t,I_0)$.



(a) Reachable set $S(t,I_0)$. (b) Over-approximation set $\overline{S}(t,I_0)$.

Figure 3.5: Concept of over-approximation.

By using over-approximation in reachability analysis, there can be instances where a system is incorrectly assessed as capable of reaching bad states. In the case of Figure 3.6a, the reachable set $S(t,I_0)$ reaches bad states. In Figure 3.6b on the other side, the over-approximation set $\overline{S}$ reaches the bad states, while the actual system is still safe. If the over-approximation does not include any bad states, then it is certain that the original reachable state set is also safe. For this reason, the over-approximative calculations are an optimal approach to safety verification.



(a) Reachable set $S(t,I_0)$ reaches bad states (red box). (b) Over-approximation set $\overline{S}(t,I_0)$ reaches bad states (red box).

Figure 3.6: Over-approximation reaching bad states.

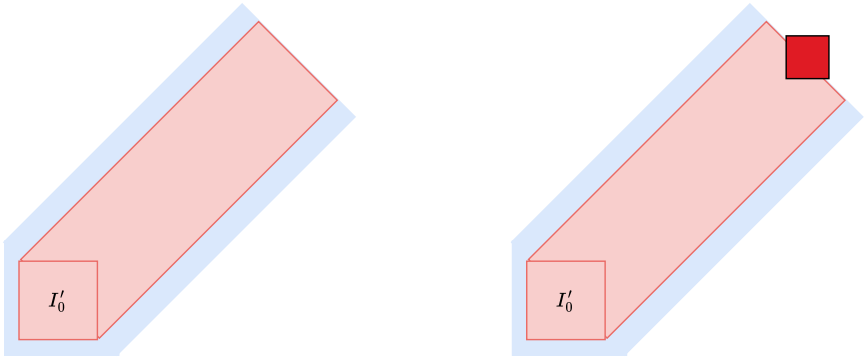
In contrast to over-approximation, where we estimate a larger set that includes all possible reachable states and some extra ones, under-approximation focuses on a smaller set. This smaller set only includes states, where we are sure that they can be

reached.

Definition 3.3.3 (Under-Approximation)**.** Given a reachable set $S(t,I_0)$, where $t \in \mathbb{R}^+$ is the time and $I_0 \subseteq \mathbb{R}^d$ for some $d \in \mathbb{N}$ is the initial set of states, the under-approximation set $\underline{S}(t,I_0)$ is defined as a non-empty set [16]:

$$\underline{S}(t,I_0) \subseteq S(t,I_0)$$

Considering the example from before in Figure 3.5a, an under-approximation (red area) of the reachable state set (blue area) is shown in Figure 3.7a. The set $\underline{S}(t,I_0)$ contains fewer states than the original reachable set $S(t,I_0)$, but the under-approximation ensures that these states are reachable. In case the under-approximation set reaches a bad state, then it is certain that the original reachable state set also can reach a bad state, as shown in Figure 3.7b.



(a) Under-approximation set $\underline{S}(t,I_0')$.

(b) Under-approximation set $\underline{S}(t,I_0')$ reaches bad states (red box).

Figure 3.7: Concept of under-approximation.

Chapter 4

HyPro Library

HyPro is a C++ library that was introduced in 2017 and is continuously being improved and extended [12]. It provides utility tools for reachability analysis of hybrid automata, including those based on flowpipe construction. Additionally, HyPro supports a variety of automaton types, such as rectangular automata, and offers methods for calculating reachable states even in the absence of flowpipe construction. This flexibility allows it to handle different modeling approaches and analysis needs in hybrid systems. It presents a variety of commonly employed state set representations such as Boxes [13], H-Polytopes, and V-Polytopes, all unified under a shared interface. HyPro not only offers a variety of state set representations but also provides a diverse range of algorithms for reachability analysis across different subclasses of hybrid automata. Ongoing developments have led to enhancements in existing reachability analysis methods, resulting in improved efficiency and scalability. Regardless of the specific approach employed, all reachability analysis methods within HyPro conduct bounded reachability analysis. This includes incorporating upper bounds on the number of jumps taken and predefined time horizons for the successor state calculations. While other libraries like Ariadne [4] or SpaceEx [6] offer similar functionalities, HyPro stands out due to its foundation in C++ and its comprehensive suite of state set representations, along with additional methods for conversion and visualization. As an open-source library available on GitHub [1], HyPro fosters collaboration and community-driven development, making it a valuable resource for advancing the state-of-the-art in hybrid automata analysis.

In this thesis, we extend the HyPro library to enable the application of the set minus operator on two polytopes using over and under-approximation. This involves expanding the functionality of both the V- and H-Polytope classes, to support this new operator. Our new implementation is integrated into the urgent reachability analysis of hybrid automata within the HyPro library. To validate the effectiveness of our new algorithms, we create a series of examples that demonstrate their correct functionality. These examples are designed to serve as a practical illustration of how the set minus operator can be applied within the extended HyPro framework. Moreover, to provide a thorough evaluation and comparison, we develop larger and more complex models. These models are used to highlight the differences offered by our new algorithm compared to the existing setMinus-Polytopes algorithm [8]. Through

[1] https://github.com/hypro/hypro

these comparisons, we aim to highlight the improvements and potential uses of our extended features in hybrid automata reachability analysis.

Chapter 5

Computation of Set Minus Operator

In this chapter, we present the computation of the set minus operation of two convex polytopes. Computing the set minus operator using an over-approximation has the advantage of reducing the complexity of the computation, ensuring a single convex result. The focus of this thesis lies mainly on the over-approximation. Section 5.1 begins with the motivation behind the computation of the set minus operator. Subsequently, we present examples that illustrate the concept of the set minus operator. Three-dimensional examples are used to simplify the visualization of the concept for higher dimensions. We employ three different cases that show different outcomes. We formulate some additional definitions in order to use them during the next subsections, where we prove the correctness and the quality of the algorithm. In Section 5.1.4, we present the algorithm of the over-approximation as a pseudocode, bridging theory with practical application. In the following Section 5.1.5, we provide the runtime estimation of the implemented algorithm in O notation. In the last Section 5.2, we explore the idea of under-approximative computations. We provide an example of the concept and an idea of the algorithm that computes an under-approximation of the convex set minus operator.

5.1 Over-approximation

5.1.1 Motivation

A fundamental task in analyzing a hybrid automaton is to compute the set of reachable state sets that can be reached within a finite time horizon. This can be done by constructing a flowpipe as seen in Figure 3.4. Flowpipes are useful because they handle the complexity of computing reachable states in a hybrid automaton. Direct calculations of the exact reachable set are often difficult due to the non-linear dynamics of the system and the high dimensionality of the state space. The flowpipe is used to represent the set of states that a system can reach over time starting from an initial state. It breaks down the problem into smaller, more manageable segments, each representing the state space evolution over a finite time interval. This makes it possible to systematically approximate and analyze the behavior of the system. We calculate

the flowpipe segments which are over-approximations of the actual reachable states. If the intersection with the bad states is empty, we can conclude that the system is safe. In the context of this thesis, the focus lies on the case where the guards are urgent. This means that when a guard is enabled, it must be taken. Having urgent transitions in the hybrid automaton leads to a different flowpipe computation in comparison to the flowpipe of a hybrid automaton without urgent transitions.

In Figure 5.1, the black box represents the intersection of the current flowpipe segment with a guard. The arrow interprets the different urgent jumps. To compute the next flowpipe segment, the white box under the black is required. In order to calculate this, we need to compute the set minus operator of the polytope P without the subset of the polytope that satisfies the guard G.

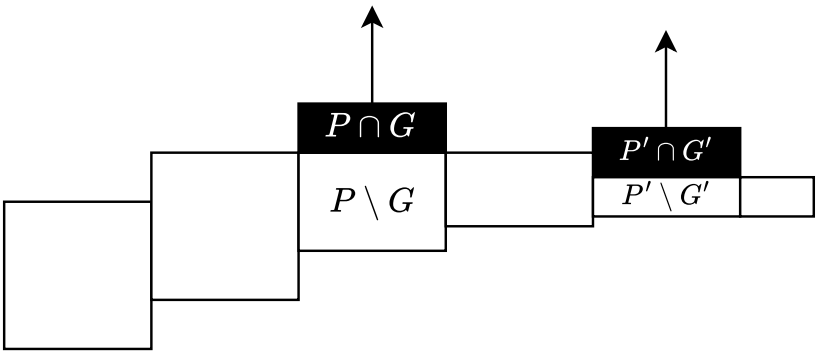
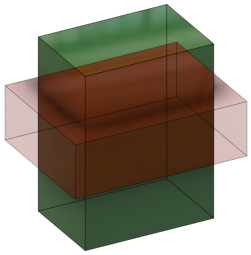


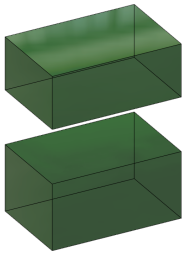
Figure 5.1: Flowpipe with urgent transitions.

In the following, we show visually how the concept of the computation of over-approximating set minus operator of two polytopes works. We present three different cases. The green polytope represents the polytope P and the red one represents the polytope G.

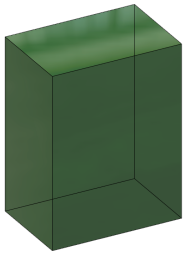
In the first example, shown in Figure 5.2a, both polytopes P and G do not contain any 0-dimensional faces that are member of the other polytope. However, there exists a shared state set of points that are member of both P and G. In Figure 5.2b, the volume set minus P without G is shown. Since the result of the volume set minus is not a single convex polytope, we need to compute the convex hull of it. The result is shown in Figure 5.2c, which is equal to the initial polytope P. As a result, the smallest resulting single convex polytope for this example can not be smaller than P.



(a) Containing the initial Polytopes P and G.



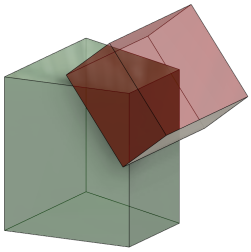
(b) Volume set minus P without G from Figure 5.2a.



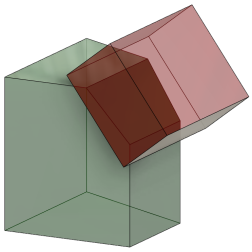
(c) Convex hull of the volume set minus P without G from Figure 5.2b.

Figure 5.2: Example 1: Visualization of a convex set minus operator given two polytopes P and G.

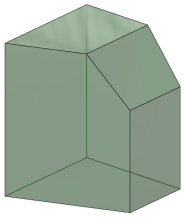
The second example, visualized in Figure 5.3a, presents the case where P has two 0-dimensional faces that are member of G. In Figure 5.3b the shared volume of P and G polytope is removed. As shown in Figure 5.3c, the remaining part is convex. As such, for this case where P has 0-dimensional faces in G, the convex set minus operator computes a polytope, which is a subset of P.



(a) Containing the initial Polytopes P and G.



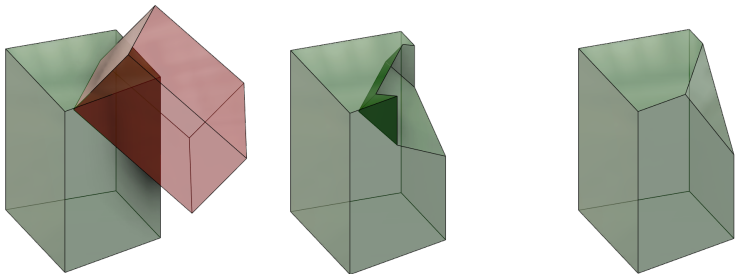
(b) Volume set minus P and G from Figure 5.3a.



(c) Convex hull of the volume set minus P without G from Figure 5.3b.

Figure 5.3: Example 2: Visualization of a convex set minus operator given two polytopes P and G.

The third example, seen in Figure 5.4a, presents the case where G has 0-dimensional faces in P and vice versa. In Figure 5.4b, the volume set minus P without G is presented. It is visible that the volume set minus polytope is not convex. This is due to the fact, that G also contains 0-dimensional faces, that are in P. By over-approximating, Figure 5.4c shows the convex hull of the volume set minus polytope. This example shows the smallest single convex polytope, that can be computed from P without G.



(a) Containing the initial Polytopes P and G.

(b) Volume set-minus P without G from Figure 5.4a.

(c) Convex hull of the volume set-minus P without G from Figure 5.4b.

Figure 5.4: Example 3: Visualization of a convex set minus operator given two polytopes P and G.

As described in Section 3, the reachability of an automaton is analyzed by constructing flowpipe segments. This allows us to explore whether the automaton reaches bad states or not. In this thesis, we focus on the development of an algorithm that computes the over-approximation set minus operators of two polytopes P and G. P represents the current flowpipe segment and G represents the guard of a transition. In Figure 5.5, we present an overview of the developed over-approximation algorithm of this thesis. The left side shows the process for the representation type V-Polytope and the right side shows the process for the representation type H-Polytope. Both functions use the main function called setMinusCrossing(P,G). This function receives as input a V-Polytope P and a polytope G. The resulting polytope of the function is in V-Representation. Given, that the reachability analysis can also be computed using H-Representation, we use conversions in combination with the function setMinusCrossing(P,G). First, the input polytope P is converted to V-Representation. This assumes, that P is bounded, as the V-Representation can only model bounded polytopes. After converting P, the setMinusCrossing can be used. The resulting polytope is then converted back to H-Representation.

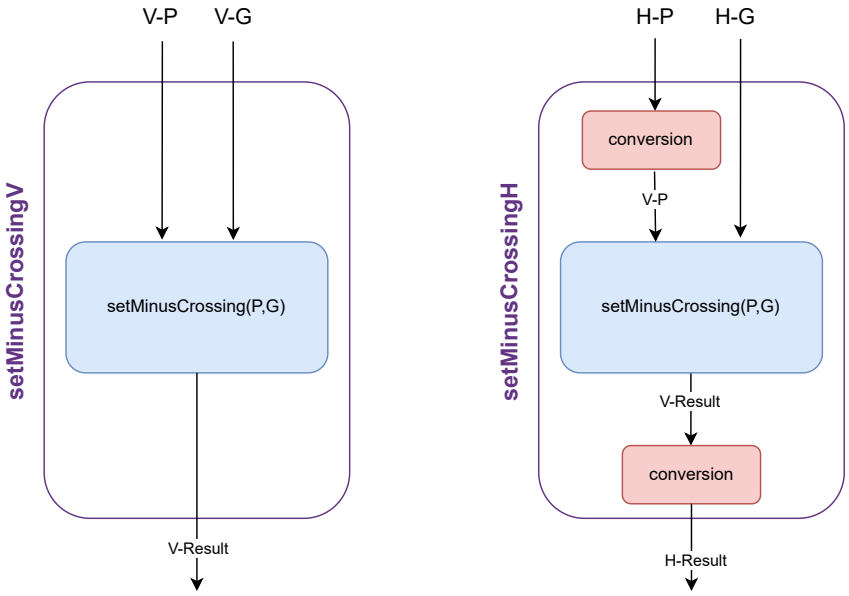


Figure 5.5: Overview of the setMinusCrossing algorithm for H- and V-Representation.

5.1.2 Definitions and Lemmas

We introduce definitions and lemmas that are necessary for proving the correctness of the algorithm. We start with basic definitions that are required for the computation of the set minus operator.

As seen in Figure 5.5, the setMinusCrossing(P,G) function computes a polytope using V-Representation. V-Polytopes are described by a set of points. However, not all points are required to define a V-Polytope. The points required for defining a V-Polytope are called extreme points:

Definition 5.1.1 (Extreme Points)**.** Let $P \in \mathbb{U}^d_{P_V}$ be a V-Polytope for some $d \in \mathbb{N}$. We call the set of 0-dimensional faces of P the set of extreme points of P.

In Figure 5.6, two V-Polytopes P (green) and G (red) are shown. The extreme points of P are extremePoints(P) = $\{p_1, p_3, p_5, p_6, p_7, p_8\}$ and the extreme points of G are extremePoints(G) = $\{g_1, g_2, g_3, g_4\}$. The points p_2 and p_4 are not extreme points of P, as they lie on the convex hull and thus they can be reconstructed from the other points.

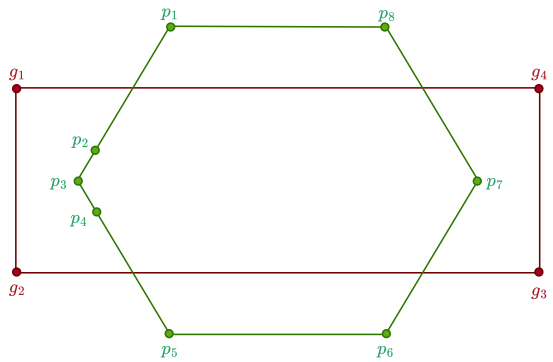


Figure 5.6: Example of extreme points of two V-Polytopes P and G.

We split the set of extreme points of a polytope into two groups, namely the pure and shared points. We define the pure points as:

Definition 5.1.2 (Pure Points)**.** Let $P,G \in \mathbb{U}_P^d$ be two polytopes for some $d \in \mathbb{N}$. We define the set of pure points of P as:

$$\text{pure}(P,G) = \{p \in \text{extremePoints}(P) \mid p \text{ not member } G\} \tag{5.1}$$

A point is member of a polytope when the conditions of Definition 3.2.7 are satisfied. The shared points are defined as follows:

Definition 5.1.3 (Shared Points)**.** Let $P,G \in \mathbb{U}_P^d$ be two polytopes for some $d \in \mathbb{N}$. The set of shared points of P is defined as:

$$\text{shared}(P,G) = \{p \in \text{extremePoints}(P) \mid p \text{ member } G\} \tag{5.2}$$

For the polytope P from Figure 5.6, the pure points are $\text{pure}(P,G) = \{p_1,p_5,p_6,p_8\}$. In the same Figure 5.6, the shared points of polytope P are $\text{shared}(P,G) = \{p_3, p_7\}$ and for G the shared points are $\text{shared}(G,P) = \{\}$.

To be able to find the minimum volume of the set minus operator and be as optimal as possible, the solution might include additional points not present in $\text{extremePoints}(P)$ and $\text{extremePoints}(G)$. We call the additional points crossing points in this thesis. The crossing points are part of the convex hull and thus over-approximate the result. To calculate them, we require a pair of points (u,u'). We begin by defining the set of connected points.

Definition 5.1.4 (Connected Points)**.** Let $P \in \mathbb{U}^d_{P_V}$ be a V-Polytope, $G \in \mathbb{U}^d_P$ be a polytope for some $d \in \mathbb{N}$ and $v \in \text{extremePoints}(P)$ be an extreme point of P.
We define the set of connected points connected as:

$$\text{connected}(v,P,G) = \{v \mid v \,\text{member}\, G\} \quad \cup X$$

where:

$$X = \bigcup_{(v,v') \in \text{1-dimensional faces}(P)} \text{connected}(v',P,G)$$

The set of connected points of a point v consists of a group of points, that are in G and share an edge (1-dimensional face) in P. This means, that for each pair of points from the connected set, there exists a sequence of edges connecting them. We calculate the set by recursively using the definition of connected on the neighboring points.
Additionally to the definition of connected points, we require a set of border vertices:

Definition 5.1.5 (Border Vertices)**.** Let $P \in \mathbb{U}^d_{P_V}$ be a V-Polytope and $G \in \mathbb{U}^d_P$ be a polytope for some $d \in \mathbb{N}$. We define the set of border vertices for an extreme point $v \in \text{extremePoints}(P)$ as:

$$\text{BV}(v,P,G) = \{p \mid (p,v) \in \text{1-dimensional faces(P)} \wedge p \text{ not member } G\} \tag{5.3}$$

To understand the concept defined from the definition of connected points and border vertices, we consider again the example from Figure 5.6. In Figure 5.6 the set of connected points of the point p_2 is $\text{connected}(p_2,P,G) = \{p_2,p_3,p_4\}$. Analogously, for the points p_3 and p_4. For the point p_7, the set of connected points is $\text{connected}(p_7,P,G) = \{p_7\}$. Given the point p_2, the border vertices are: $\text{BV}(p_2,P,G) = \{p_1\}$ and for p_7 the border vertices are $\text{BV}(p_7,P,G) = \{p_6,p_8\}$.
As described, the connected points and the border vertices allow us to calculate the set of crossing points.

Definition 5.1.6 (Crossing Points)**.** Let $P \in \mathbb{U}^d_{P_V}$ be a V-Polytope, $G \in \mathbb{U}^d_P$ be a polytope for some $d \in \mathbb{N}$ and $v \in \text{extremePoints}(P)$ be an extreme point of P.
We define the set of crossing points CP_v for v as:

$$\begin{aligned}\mathrm{CP}_v(P,G) = \{p \mid & \wedge u' \in \mathrm{connected}(v,P,G) \\ & \wedge u \in \mathrm{BV}(u',P,G) \\ & \wedge \exists \lambda \in [0,1] \subseteq \mathbb{R} : p = u + \lambda \cdot (u' - u) \\ & \wedge \nexists \lambda' \in [0,1] \subseteq \mathbb{R} : \lambda' < \lambda \wedge p = u + \lambda' \cdot (u' - u) \\ & \wedge p \,\mathrm{member}\, G\}\end{aligned} \tag{5.4}$$

Located on an edge (u,u') for two points u,u' of a polytope P, a crossing point is calculated in R^d. While u is not part of a polytope G, u' belongs to the set of connected points within G. The crossing point on the edge is the closest possible point to u while still being a member of G.
Figure 5.7 extends Figure 5.6 by containing the additional crossing points cp_1,cp_2,cp_3 and cp_4. These points are members of both P and G. As seen, the point cp_1 lies on the edge (p_1,p_2) of P, where p_2 is a connected point and p_1 is the corresponding border vertex.
In the same way, the crossing points cp_2, cp_3 and cp_4 are calculated.

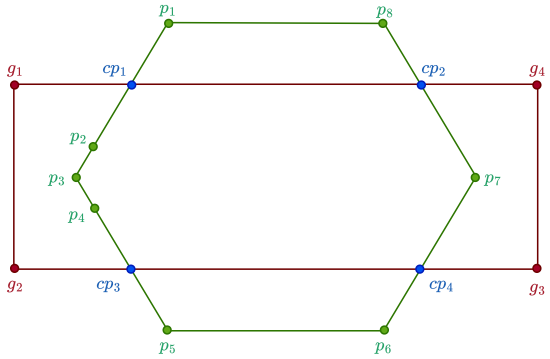


Figure 5.7: Example of crossing points of two V-Polytopes P and G.

The crossing points can be grouped with their corresponding connected points. This group represents a cutoff that will be removed from a polytope P to reduce the volume of the shared area. The mathematical definition of the cutoff is as follows:

Definition 5.1.7 (Cutoffs)**.** Let $P \in \mathbb{U}^d_{P_V}$ be a V-Polytope, $G \in \mathbb{U}^d_P$ be a polytope for some $d \in \mathbb{N}$ and $v \in \mathrm{shared}(P,G)$ be a shared point of P and G.
We define cutoff_v of v as a V-Polytope which consists of the following points:

$$\mathrm{cutoff}_v(P,G) = \text{V-P}(\mathrm{CP}_v(P,G) \cup \mathrm{connected}(v,P,G))$$

We define the set of all cutoffs as:

$$\mathrm{cutoffs}(P,G) = \{\mathrm{cutoff}_v(P,G) \mid v \in \mathrm{shared}(P,G)\} \tag{5.5}$$

For a shared point v from two polytopes P and G, its corresponding cutoff is described as a V-Polytope. The set cutoffs contains all cutoffs from two given polytopes P and G.
In Figure 5.8, there are two cutoff polytopes present. One cutoff polytope is cutoff_1 = V-P($\{cp_1,p_2,p_3,p_4,cp_3\}$) and the other one is cutoff_2 = V-P($\{cp_2,p_7,cp_4\}$). The set containing all the cutoff polytopes is cutoffs = $\{\text{cutoff}_1, \text{cutoff}_2\}$.
Highlighted in green is the volume of the cutoffs polytopes cutoff_1 and cutoff_2.

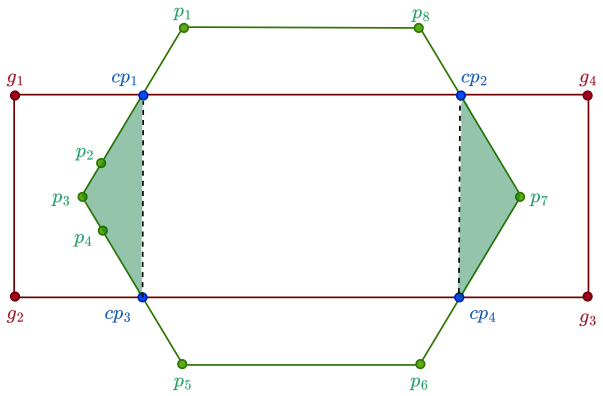


Figure 5.8: Example of two V-Polytopes P and G with their corresponding cutoffs.

Next, we introduce some definitions of operators. Given two polytopes, there is a need to compare them and apply operations on them. We start by introducing an operator that validates whether one polytope is fully included in the other.

Definition 5.1.8 (Polytope Inclusion)**.** Let $A,B \in \mathbb{U}_P^d$ be two polytopes for some $d \in \mathbb{N}$.
We say that A is included in B if and only if:

$$\forall_{x \in \mathbb{R}^d} \quad x \text{ member } A \rightarrow x \text{ member } B \tag{5.6}$$

We denote this as $A \leq B$.

In Definition 5.1.8, we do not use $\subseteq$ on purpose since we compare the volume of two polytopes and not the set of extreme points that define the polytopes in V-Representation.
Instead of focusing only on convex polytopes, we aim to describe the combination of two polytopes, which do not necessarily have to be convex. To combine them, we use the following definition:

Definition 5.1.9 (Combined Polytopes)**.** Let A,$B \in \mathbb{U}_P^d$ be two polytopes for some $d \in \mathbb{N}$.
We define the operator "$\cup$" as the volume addition of two polytopes with:

$$A \cup B = \{x \in \mathbb{R}^d \mid x \,\text{member}\, A \vee x \,\text{member}\, B\} \tag{5.7}$$

The operator "$\cup$" allows us to combine two polytopes and define all points that are in either one of the polytopes. It is worth mentioning that the operator "$\cup$" can not be used in general to combine the extreme points of two polytopes, as the result would not be the exact volume of the polytopes.
To validate the convex set minus to be a correct over-approximation, we require a ground truth, which must be included in the over-approximation. The ground truth for the set minus of two polytopes A and B corresponds to the volume set minus defined as follows:

Definition 5.1.10 (Volume Set Minus)**.** Let $P \in \mathbb{U}_{P_V}^d$ be a V-Polytope and $G \in \mathbb{U}_P^d$ be a polytope for some $d \in \mathbb{N}$.
We define the volume set minus SM of P without G as:

$$\text{SM}(P,G) = \{p \in \mathbb{R}^d \mid p \,\text{member}\, P \wedge \neg(p \,\text{member}\, G)\} \tag{5.8}$$

With Definition 5.1.10, we can describe specifically which state set must be part of the convex set minus over-approximation.
In Figure 5.9, the red area indicates the volume set minus of the polytope P without G. The volume set minus consists of all points that are only member of P and not member of G.

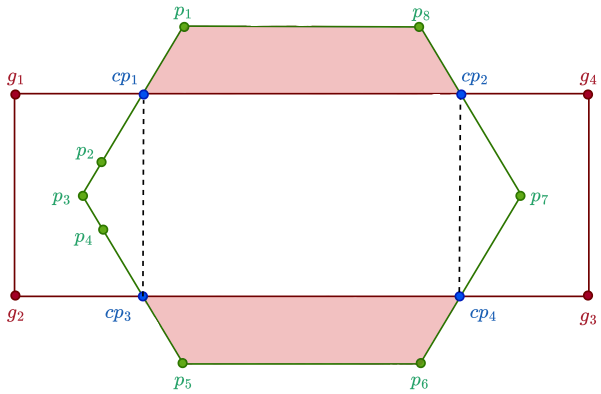


Figure 5.9: Red area shows the volume set minus of P without G.

After formulating the relation $\leq$, the operator "$\cup$", and the volume set minus SM,

we now present the mathematical definition of our convex set minus as an over-approximation:

Definition 5.1.11 (Convex Set Minus)**.** Let $P \in \mathbb{U}^d_{P_V}$ be a V-Polytope and $G \in \mathbb{U}^d_P$ be a polyhedron for some $d \in \mathbb{N}$.
We define the convex set minus $\overline{\text{SM}}$ of P without G as a V-Polytope:

$$\overline{\text{SM}}(P,G) = \text{V-P}\left(\text{pure}(P,G) \cup \bigcup_{v \in \text{shared}(P,G)} \text{CP}_v(P,G)\right) \tag{5.9}$$

The convex set minus as an over-approximation consists of the set of pure points with crossing points.
In Figure 5.10, the blue area highlights the convex set minus of P without G. We can see, that the blue area includes all points from the volume set minus, shown in Figure 5.9.

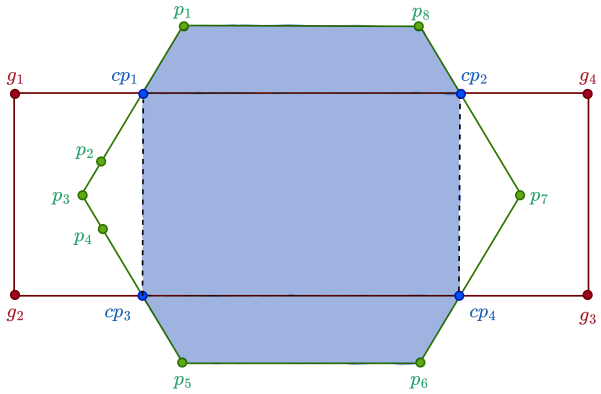


Figure 5.10: Blue area shows the convex set minus of P without G.

In order to show that $\overline{SM}$ is a valid over-approximation of the set minus operator, we require a lemma that combines the $\overline{SM}$ with a cutoff polytope. In simple words, we show that the original polytope P can be reconstructed by combining $\overline{SM}$ with all cutoff polytopes.

Lemma 5.1.12 (Combination of Polytopes)**.** Let $P \in \mathbb{U}^d_{P_V}$ be a V-Polytope, $G \in \mathbb{U}^d_P$ be a polytope for some $d \in \mathbb{N}$ and let $\text{cutoff}_v(P,G)$ be the cutoff of a shared point v of P and G.
The following holds:

$$\overline{\text{SM}}(P,G) \cup \text{cutoff}_v(P,G) = \text{V-P}(X) \tag{5.10}$$

where:

$$X = \text{pure}(P,G) \cup \bigcup_{v \in \text{shared}(P,G)} \text{CP}_v(P,G) \cup \text{connected}(v,P,G)$$

Proof: From the Definitions 5.1.11 and 5.1.7 it holds:

$$\text{cutoff}_v(P,G) \supseteq \text{CP}_v(P,G) \subseteq \overline{\text{SM}}(P,G) \tag{5.11}$$

Given a shared point v of P and G, each crossing point of v is constructed from two points $u,u' \in \mathbb{R}^d$ with $u \in \overline{\text{SM}}(P,G)$ and $u' \in \text{cutoff}_v(P,G)$. We know that each crossing point lies on the edge $(u,u') \in$ 1-dimensional faces of P and that P is convex. With Equation 5.11, where it holds that the crossing points are part of the cutoffs and $\overline{SM}$, we can conclude that the combination of $\overline{SM}$ and cutoff_v will result in the convex polytope:

$$\text{V-P}\left(\text{pure}(P,G) \cup \bigcup_{v \in \text{shared}(P,G)} \text{CP}_v(P,G) \cup \text{connected}(v,P,G)\right)$$

Lemma 5.1.13 (Cutoffs are part of polytope G)**.** Let $P \in \mathbb{U}^d_{P_V}$ be a V-Polyhedron, $G \in \mathbb{U}^d_P$ be a polyhedron for some $d \in \mathbb{N}$ and let $v \in \text{shared}(P,G)$ be a shared point of P and G.
The following holds:

$$\text{cutoff}_v(P,G) \leq G \tag{5.12}$$

Proof: From Definition 5.1.7 it holds that a cutoff consists of the connected points $\text{connected}(v,P,G)$ and the crossing points $\text{CP}_v(P,G)$ given a shared point v of P and G. From Definition 5.1.4 it holds that the connected points are part of G. Similar to the connected points, the crossing points are part of G as well. Since all points of $\text{cutoff}_v(P,G)$ are part of G, it follows that $\text{cutoff}_v(P,G) \leq G$.

Lemma 5.1.14 (Construction of P from $\overline{SM}(P,G)$ and cutoffs)**.** Let $P \in \mathbb{U}^d_{P_V}$ be a V-Polytope and $G \in \mathbb{U}^d_P$ be a polyhedron for some $d \in \mathbb{N}$.
The following holds:

$$\overline{\text{SM}}(P,G) \cup \bigcup_{v \in \text{shared}(P,G)} \text{cutoff}_v(P,G) = P \tag{5.13}$$

Proof:

$$\overline{\text{SM}}(P,G) \cup \textstyle\bigcup_{v \in \text{shared}(P,G)} \text{cutoff}_v(P,G) \leq P$$

Given that P is convex we show that for all extreme points of $\overline{\text{SM}}(P,G)$ and for each polytope in cutoffs(P,G) it holds that they are inside P. For $\overline{\text{SM}}(P,G)$ which consists of the pure points and the crossing points, it holds that the pure points are member of P, given Definition 5.1.2. The crossing points are member of P as well since they are constructed from two points u,$u' \in \mathbb{R}^d$ with u,$u' \in$ extremePoints(P). Given that each crossing point lies on the edge (u,u'), P is convex, and that u and u' are extreme points of P means that the crossing points are also member of P. The cutoff polytopes are constructed from the connected points and the crossing points. The connected points are extreme points of P, since they lie on the 1-dimensional faces of P. We showed above that the crossing points are part of P. This means that each cutoff must be within P as well. As a result, the combined polytope $\overline{\text{SM}}(P,G)$ and the cutoffs are part of P.

Next, we show the opposite direction of the inequality:

$$\overline{\text{SM}}(P,G) \cup \bigcup_{v \in \text{shared}(P,G)} \text{cutoff}_v(P,G) \geq P$$

We show that all extreme points of P are part of the volume addition of $\overline{\text{SM}}(P,G)$ and the cutoffs. For that, we combine the polytope $\overline{\text{SM}}(P,G)$ and the cutoffs by using Lemma 5.1.12:

$$\overline{\text{SM}}(P,G) \cup \bigcup_{v \in \text{shared}(P,G)} \text{cutoff}_v(P,G) = \text{V-P}(X) \tag{5.14}$$

where:

$$X = \text{pure}(P,G) \cup \bigcup_{v \in \text{shared}(P,G)} \text{CP}_v(P,G) \cup \bigcup_{v \in \text{shared}(P,G)} \text{connected}(v,P,G)$$

As seen in Equation 5.14, the combined polytope consists of the pure points of P, the crossing points and the connected points of each cutoff. We know that all extreme points of P are either pure points or shared points. The pure points are part of the combined polytope, given Equation 5.14. From Definition 5.1.4, we know that the connected points are constructed from the shared points. As such, the combined polytope contains all extreme points of P and is therefore greater or equal to P.
We showed:

$$\overline{\text{SM}}(P,G) \cup \bigcup_{v \in \text{shared}(P,G)} \text{cutoff}_v(P,G) \leq P$$

$$\overline{\text{SM}}(P,G) \cup \bigcup_{v \in \text{shared}(P,G)} \text{cutoff}_v(P,G) \geq P$$

As such, it follows:

$$\overline{\text{SM}}(P,G) \cup \bigcup_{v \in \text{shared}(P,G)} \text{cutoff}_v(P,G) = P$$

5.1.3 Proof of Optimality

In this section, we present two proofs for the convex set minus over-approximation. We begin with the proof of correctness, which means that the convex set minus contains the set minus volume of the polytopes P without G.

$$\mathrm{SM}(P,G) \leq \overline{\mathrm{SM}}(P,G) \tag{5.15}$$

The proof uses Lemmas 5.1.13 and 5.1.14.
Proof by contradiction: If the convex set minus does not contain the set minus volume of the polytopes P without G, then there exists a point p that is member of $\mathrm{SM}(P,G)$ but not in $\overline{\mathrm{SM}}(P,G)$. From Lemma 5.1.14 we know that $\overline{\mathrm{SM}}(P,G)$ combined with the cutoffs is equal to P. We know that $\mathrm{SM}(P,G) \leq P$. Since p is a member of $\mathrm{SM}(P,G)$ it means that p must be a member of P. If the point p is not in $\overline{\mathrm{SM}}(P,G)$, then it must be in one cutoff polytope. From Lemma 5.1.13 we know that the cutoffs are part of G. This means that the point p is a member of G. This is a contradiction since the point p is in $\mathrm{SM}(P,G)$ and thus cannot be in G.

The second proof shows that the result of the convex set minus $\overline{\mathrm{SM}}(P,G)$ contains the smallest convex set that includes the volume set minus $\mathrm{SM}(P,G)$.
Proof by contradiction: If $\overline{\mathrm{SM}}(P,G)$ is not minimal, then there exists a convex polytope $\overline{\mathrm{SM}}'(P,G)$ which contains $\mathrm{SM}(P,G)$ and for which it holds:

$$\exists p \in \mathrm{extremePoints}(\overline{\mathrm{SM}}(\mathrm{P,G})) : \neg(p\,\mathrm{member}\,\overline{\mathrm{SM}}'(P,G)) \tag{5.16}$$

Equation 5.16 states that there exists an extreme point p of $\overline{\mathrm{SM}}(P,G)$ that is not part of $\overline{\mathrm{SM}}'(P,G)$. The extreme points of $\overline{\mathrm{SM}}(P,G)$ consist of the pure points of P and the crossing points. The pure points from $\overline{\mathrm{SM}}$ must be part of $\overline{\mathrm{SM}}'(P,G)$, since the pure points are members of P without G and it holds $\mathrm{SM}(P,G) \leq \overline{\mathrm{SM}}'(P,G)$. The crossing points of $\overline{\mathrm{SM}}$ are constructed from two points $u,u' \in \mathbb{R}^d$ with $u' \in \mathrm{connected}(v,P,G)$ and $u \in \mathrm{BV}(u',P,G)$ for any shared point v. The crossing points are constructed by locating the closest point to u and is still a member of G. As such, the crossing points are part of $\overline{\mathrm{SM}}'(P,G)$ as well since there is no closer point to u that is a member of G. Given that all extreme points of $\overline{\mathrm{SM}}$ are part of $\overline{\mathrm{SM}}'$, means that there cannot exist an extreme point p that is not part of $\overline{\mathrm{SM}}'(P,G)$. Therefore, $\overline{\mathrm{SM}}(P,G)$ is minimal.

5.1.4 Algorithm

In this section, the pseudocode for the convex set minus operator of two polytopes as an over-approximation is presented. Firstly, we start with the main algorithm and subsequently, we present helper functions that are used by the main algorithm. While the above-mentioned proof and the main algorithm rely on points and thus on the V-Representation, via conversion, the algorithm is also possible to handle H-Polytopes. Algorithm 2 shows the pseudocode of the main algorithm. In the first line, we compute the extreme points of a polytope P. After obtaining the extreme points, we compute the 1-dimensional faces of P with the help of the function getConvexEdges. The 1-dimensional faces are stored in the variable edgesP. In the next line, we determine the extreme points of P that are in G and store them in PnG. We do that by using the requirements of membership mentioned in Definition 3.2.7. We iterate over the points of PnG and for each of them we compute their corresponding border vertices that are in P. After calculating the border vertices, we iterate over them to derive the crossing points. After calculating the crossing points, the algorithm returns a V-Polytope consisting of the pure points of P together with the crossing points that were computed.

Algorithm 2 SetMinusCrossing(P,G)

Input: V-Polytope P, Polytope G
Output: V-Polytope($P \setminus G$)

```
1: extremePoints = getExtremePoints(P)                          ▷ Algo. 5
2: edgesP = getConvexEdges(extremePoints)                       ▷ Algo. 6
3: PnG = {v ∈ extremePoints | v member G}
4: for v ∈ PnG do
5:     BVs = getBorderVertices(v,edgesP, PnG)                   ▷ Algo. 7
6:     for Point v' ∈ BVs do
7:         if G is V-Polytope then
8:             CPs = CPs ∪ crossingPointsV(v,v',P,G)            ▷ Algo. 3
9:         else
10:            CPs = CPs ∪ crossingPointsH(v,v',P,G)            ▷ Algo. 4
11:        end if
12:    end for
13: end for
14: return (P \ PnG) ∪ CPs
```

For the derivation of the crossing points, we differentiate between two cases. These cases are shown in Algorithm 3 and Algorithm 4. The first case is when G is bounded and in V-Representation. To find the crossing points, we use linear programming to find a λ' that satisfies the membership requirement in G, and that is also part of a crossing edge $v' + \lambda' \cdot (v - v')$. Considering that the point v' is the border vertex of the point v, we try to find the closest point to v', in order to be as optimal as possible. To achieve that, we minimize λ' to find the closest crossing point to v'.

Algorithm 3 crossingPointsV

Input: Point v, Point v', V-Polytope P, V-Polytope G
Output: CP

1: Points $w_1, \cdots, w_n = G$.getVertices()
2: // Calculate the closest point that intersects with a facet via LP solver:

$$\begin{aligned} \min_{\lambda_1,\ldots,\lambda_n,\lambda' \in [0,1] \subseteq \mathbb{R}} \quad & \lambda' \\ \text{s.t.} \quad & \sum_{i=1}^{n} \lambda_i = 1 \\ & \underbrace{\sum_{i=1}^{n} \lambda_i \cdot w_i}_{\text{Member 3.2.7}} = \underbrace{v' + \lambda' \cdot (v - v')}_{\text{Crossing Edge}} \end{aligned}$$

3: CP = $\{v' + \lambda' \cdot (v - v')\}$
4: **return** CP

In the case where G is in H-Representation, we use the constraints of G to find the crossing point. For this case, we store the constraints that define G in the form of pairs, where a_i indicates a vector of coefficients of each constraint and b is the

scalar. Again, we use linear programming to find the minimum λ' that satisfies all the constraints.

Algorithm 4 crossingPointsH

Input: Point v, Point v',V-Polytope P, H-Polytope G
Output: CP

1: Constraints $(\mathbf{a}_1,b_1),\cdots,(\mathbf{a}_m,b_m) = G$.getConstraints()
2: // Calculate the closest point that fulfills all constraints:

$$\min_{\lambda' \in [0,1] \subseteq \mathbb{R}} \quad \lambda'$$
$$\text{s.t.} \quad \forall_{1 \leq i \leq m} \quad \mathbf{a}_i \cdot \underbrace{\left(v' + \lambda' \cdot (v - v')\right)}_{\text{Point}} \leq b_i$$

3: CP $= \{v' + \lambda' \cdot (v - v')\}$
4: **return** CP

What remains unclear until now, is how we compute the extreme points and the 1-dimensional faces of a polytope. Algorithm 5 visualizes as pseudocode the calculation of the extreme points from a given V-Polytope P. In the beginning, we store all the unique points of P. The algorithm then iterates over all unique points of P. Using linear programming, the algorithm evaluates whether or not the current point can be reconstructed from a combination of all the other points. If this is the case, then there exists a solution and thus the point is not an extreme point of P. If the other points can not reconstruct the current point, then a solution does not exist when solving the linear programming and thus the point is an extreme point of P. The algorithm inserts all extreme points into the variable ExtremePoints and returns it.

Algorithm 5 getExtremePoints

Input: V-Polytope $P = \{p_1,\cdots,p_n\}$
Output: extremePoints(P)

1: extremePoints $= \emptyset$
2: uniquePoints $= P$.getUniquePoints() // $\{p_1,\ldots,p_m\}$
3: **for** Point $p_i \in$ uniquePoints **do**

$$\min_{\lambda_1,\ldots,\lambda_m \in [0,1] \subseteq \mathbb{R}} \quad 0$$
$$\text{s.t.} \quad \sum_{k \neq i}^{m} \lambda_k \cdot p_k = p_i \wedge \sum_{k \neq i}^{m} \lambda_k = 1$$

4: **if** $\nexists \lambda_1,\ldots,\lambda_m$ **then**
5: extremePoints $=$ extremePoints $\cup \{p_i\}$
6: **end if**
7: **end for**
8: **return** extremePoints

Algorithm 6 visualizes a method for obtaining the 1-dimensional faces of a polytope P. Given the extreme points of P, the algorithm iterates over all pairs of points of

P. For each pair, linear programming is used. If a point on the edge between the two selected points can be reconstructed from all other points of P, then the pair representing an edge is not part of the 1-dimensional faces.

Algorithm 6 getConvexEdges

Input: extremePoints(P) = $\{p_1, \cdots, p_n\}$
Output: Convex edges of P

1: convexEdges = $\emptyset$
2: **for** Point $p_i \in$ extremePoints(P) **do**
3: **for** Point $p_j \in$ extremePoints(P) **do**
4: **if** $p_i \neq p_j$ **then**

$$\min_{\lambda_1,\ldots,\lambda_n \in [0,1] \subseteq \mathbb{R}} \quad 0$$
$$\text{s.t.} \quad \sum_{k \neq i,j}^{n} \lambda_k \cdot p_k = \lambda_i \cdot p_i + \lambda_j \cdot p_j$$
$$\sum_{k \neq i,j}^{n} \lambda_k = 1 \wedge \lambda_i + \lambda_j = 1$$

5: **if** $\nexists \lambda_1, \ldots, \lambda_n$ **then**
6: convexEdges = convexEdges $\cup \{(p_i,p_j)\}$
7: **end if**
8: **end if**
9: **end for**
10: **end for**
11: **return** convexEdges

The computation of the border vertices is shown in Algorithm 7. The algorithm receives as an input a point v, a set of 1-dimensional faces, and the set PnG. The algorithm iterates over all edges. For each edge, the algorithm checks if v is part of it and if their direct neighbor is not in the set PnG. If that is the case, then the neighboring point is inserted in the set BV. In the end, the algorithm returns a set that contains the border vertices of the given point v.

Algorithm 7 getBorderVertices

Input: Point v, edgesP = $[e_1,\ldots,e_n]$, PnG = $\{v_1,\ldots,v_k\}$
Output: Border vertices of v

1: BVs = $\emptyset$
2: **for** $(p,p') \in$ edgesP **do**
3: **if** $v = p \wedge p' \notin$ PnG **then**
4: BVs = BVs $\cup \{p'\}$
5: **end if**
6: **if** $v = p' \wedge p \notin$ PnG **then**
7: BVs = BVs $\cup \{p\}$
8: **end if**
9: **end for**
10: **return** BVs

The reachability analysis in HyPro uses a representation type, which defines the input and output of the analysis. This means that the inputs and the result are calculated and represented in a single representation type.

The two Algorithms 8 and 9 integrate these two cases. Algorithm 8 handles the case where the input and output are in V-Representation. The algorithm calls Algorithm 2 and returns the result.

Algorithm 8 setMinusCrossingV

Input: V-Polytope P, V-Polytope G
Output: V-Polytope($P \setminus G$)

1: **return** setMinusCrossing(P,G)

Algorithm 9 receives as input two H-Polytopes and returns the H-Polytope that is the result of the convex set minus operation. The algorithm requires that the polytope P is bounded. First, we convert the polytope P to V-Polytope. After that, we call Algorithm 2, which receives a V-Polytope P and an H-Polytope G as input. After calculating the result, Algorithm 9 converts the resulting V-Polytope to H-Polytope and returns it.

Algorithm 9 setMinusCrossingH

Input: H-Polytope P, H-Polytope G
Output: H-Polytope($P \setminus G$)

1: $\mathrm{P_V}$ = toVPolytope(P)
2: $\mathrm{res_V}$ = $\mathrm{P_V}$.setMinusCrossing(G)
3: $\mathrm{res_H}$ = toHPolytope(polytope)
4: **return** $\mathrm{res_H}$

5.1.5 Runtime

The runtime of an algorithm is a critical factor in determining its efficiency and practicality. In this section, we analyze the theoretical runtime of our proposed Algorithm 2 and evaluate its performance. First, we calculate the runtime of the helper functions and then conclude with the runtime of the main algorithm.

In Tables 5.11 and 5.12, we define variables with a short description that we use to estimate the runtime of the algorithms.

Symbol	Description
k	dimension = $x_0, \cdots, x_k$
n_1	number of points in P
n_2	number of extreme points in P
m	number of points in G
s	number of extreme points of P in G

Figure 5.11: Table for the description of the variables used to estimate the runtime.

LP Runtime Estimation = $\mathcal{O}(i_1 \cdot v \cdot c)$	
Symbol	Description
i_1	number of iterations to find the best solution using LP
i_2	number of iterations to find any solution using LP
v	number of variables
c	number of constraints

Figure 5.12: Table for the description of the variables used to estimate LP runtime.

First, we will analyze the runtime of Algorithm 5. This function is responsible for computing the extreme points of a V-Polytope. To find the extreme points, first, we have to calculate the unique points of the V-Polytope. While there might exist fast methods, the pairwise comparison of each point has a runtime of n_1^2. In the worst case, each point in the V-Polytope is a unique point. As a result, we consider this worst-case scenario, where the number of unique points is equal to the number of points in the V-Polytope. We then use linear programming for each unique point. It is known, that the runtime of LP, for example using simplex, has time complexity $\mathcal{O}(i_1 \cdot v \cdot c)$. Since we have n points and for each point, we need to use LP, the runtime of Algorithm 5 is calculated as follows:

$$\begin{aligned} \text{getExtremePointsRuntime} = \quad & \mathcal{O}_{getEP}(n_1^2 + n_1 \cdot \underbrace{i_2 \cdot v \cdot c}_{LP}) \\ \Rightarrow \quad & \mathcal{O}_{getEP}(n_1^2 + n_1 \cdot i_2 \cdot n_1 \cdot 2) \\ \Rightarrow \quad & \mathcal{O}_{getEP}(n_1^2 + n_1^2 \cdot i_2) \\ \Rightarrow \quad & \mathcal{O}_{getEP}(n_1^2 \cdot i_2) \end{aligned}$$

The next function that we analyze is the runtime of Algorithm 6. This function is responsible for computing the convex edges of a V-Polytope. In this function, we have two nested loops that iterate over all extreme points of P. For each pair of points, we use linear programming to find the convex edges. The runtime of the Algorithm 6 is calculated as follows:

$$\begin{aligned} \text{getConvexEdgesRuntime} = \quad & \mathcal{O}_{getCE}(n_2 \cdot n_2 \cdot \underbrace{i_2 \cdot v \cdot c}_{LP}) \\ \Rightarrow \quad & \mathcal{O}_{getCE}(n_2^2 \cdot i_2 \cdot n_1 \cdot 3) \\ \Rightarrow \quad & \mathcal{O}_{getCE}(n_2^2 \cdot n_1 \cdot i_2) \end{aligned}$$

The runtime of the helper functions getExtremePoints and getConvexEdges are calculated above. In the next step, we analyze the runtime of the membership function. This function is responsible for checking if a point is a member of a polytope. In our case, in the polytope G. This is done again with linear programming, thus the runtime of the membership function is:

$$\begin{aligned} \text{membershipRuntime} = \quad & \mathcal{O}_{member_G}(\underbrace{i_2 \cdot v \cdot c}_{LP}) \\ \Rightarrow \quad & \mathcal{O}_{member_G}(i_2 \cdot m \cdot 2) \\ \Rightarrow \quad & \mathcal{O}_{member_G}(i_2 \cdot m) \end{aligned}$$

The next step is to compute the runtime of the border vertices of a point. We consider the worst case, where each point in the set PnG has two border vertices. The runtime of the Algorithm 7 is calculated as follows:

$$\begin{aligned} \text{getBorderVerticesRuntime} = \quad & \mathcal{O}_{getBV}(2 \cdot s) \\ \Rightarrow \quad & \mathcal{O}_{getBV}(s) \end{aligned}$$

In Algorithm 2, we have to calculate the crossing points. We have two cases, one where G is in V-Representation and the other where it is in H-Representation. In general, the runtime of calculating the crossing points with G in V-Representation is higher than with G in H-Representation. This is because in V-Representation, calculating whether a point is in G is computationally more expensive than in H-Representation. The runtime of the crossing points from lines 8-11 from Algorithm 2 is calculated as follows:

$$\begin{aligned} \text{getCrossingPointsRuntime} = \quad & \mathcal{O}_{getCP}(\underbrace{i_1 \cdot v \cdot c}_{LP}) \\ \Rightarrow \quad & \mathcal{O}_{getCP}(i_1 \cdot (m+1) \cdot 2) \\ \Rightarrow \quad & \mathcal{O}_{getCP}(i_1 \cdot (m+1)) \\ \Rightarrow \quad & \mathcal{O}_{getCP}(i_1 \cdot m + i_1) \\ \Rightarrow \quad & \mathcal{O}_{getCP}(i_1 \cdot m) \end{aligned}$$

After calculating the runtime of all the helper functions, we can now calculate the runtime of the main algorithm. The runtime of the Algorithm 2 is calculated as follows:

$$
\begin{aligned}
\text{Runtime } 2 = \quad & \mathcal{O}_{getEP} + \mathcal{O}_{getCE} + n_2 \cdot \mathcal{O}_{member_G} \\
& + s \cdot \mathcal{O}_{getBV} + s \cdot \mathcal{O}_{getBV} \cdot \mathcal{O}_{getCP} \\
\Rightarrow \quad & \mathcal{O}(n_1^2 \cdot i_2) + \mathcal{O}(n_2^2 \cdot n_1 \cdot i_2) + n_2 \cdot \mathcal{O}(i_2 \cdot m) \\
& + s \cdot \mathcal{O}(s) + s \cdot \mathcal{O}(s) \cdot \mathcal{O}(i_1 \cdot m) \\
\Rightarrow \quad & \mathcal{O}(n_1^2 \cdot i_2) + \mathcal{O}(n_2^2 \cdot n_1 \cdot i_2) + \mathcal{O}(n_2 \cdot i_2 \cdot m) \\
& + \mathcal{O}(s^2) + \mathcal{O}(s^2 \cdot i_1 \cdot m) \qquad | \; s \leq n_1, n_2 \leq n_1 \\
\Rightarrow \quad & \mathcal{O}(n_1^2 \cdot i_2) + \mathcal{O}(n_1^2 \cdot n_1 \cdot i_2) + \mathcal{O}(n_1 \cdot i_2 \cdot m) \\
& + \mathcal{O}(n_1^2) + \mathcal{O}(n_1^2 \cdot i_1 \cdot m) \\
\Rightarrow \quad & \mathcal{O}(n_1^2 \cdot i_2) + \mathcal{O}(n_1^3 \cdot i_2) + \mathcal{O}(n_1 \cdot i_2 \cdot m) \\
& + \mathcal{O}(n_1^2) + \mathcal{O}(n_1^2 \cdot i_1 \cdot m) \\
\Rightarrow \quad & \mathcal{O}(n_1^3 \cdot i_2) + \mathcal{O}(n_1 \cdot i_2 \cdot m) + \mathcal{O}(n_1^2 \cdot i_1 \cdot m) \qquad (5.17)
\end{aligned}
$$

Assumption:

- $i_1 =$ find best solution $\in \mathcal{O}(n^3)$ for input size n
- $i_2 =$ find any solution $\in \mathcal{O}(n^2)$ for input size n

$$
\begin{aligned}
\text{Runtime } 2 = \quad & \mathcal{O}(\underbrace{n_1^3 \cdot i_2}_{getCE}) + \mathcal{O}(\underbrace{n_1 \cdot i_2 \cdot m}_{member_G}) + \mathcal{O}(\underbrace{n_1^2 \cdot i_1 \cdot m}_{getCP}) \qquad \text{using (5.17)} \\
\Rightarrow \quad & \mathcal{O}(n_1^3 \cdot n_1^2) + \mathcal{O}(n_1 \cdot m^2 \cdot m) + \mathcal{O}(n_1^2 \cdot m^3 \cdot m) \qquad (5.18) \\
\Rightarrow \quad & \mathcal{O}(n_1^5) + \mathcal{O}(n_1 \cdot m^3) + \mathcal{O}(n_1^2 \cdot m^4) \\
\Rightarrow \quad & \mathcal{O}(n_1^5) + \mathcal{O}(n_1^2 \cdot m^4) \qquad (5.19)
\end{aligned}
$$

We know that LP in the worst case can take an exponential amount of time to solve. However, in practice, the LP method is very efficient and can solve most problems in polynomial time. We assume that for i_2, which is finding any solution with LP, the runtime is in $\mathcal{O}(n^2)$. For i_1, which is finding the best solution with LP, we assume a runtime in $\mathcal{O}(n^3)$. In Equation 5.18, we replace the variables with the assumptions. We replace i_1 and i_2 with the variables n_1 and m depending on the input size of the LP method. The runtime of Algorithm 2 is shown in Equation 5.19. Given the runtime and that n_1 is the number of points in the V-Polytope P and m is the number of points in the polytope G, we can conclude that the runtime of Algorithm 2 increases with n_1^5 depending on P and with $n_1^2 \cdot m^4$ depending on G.

5.2 Under-approximation

5.2.1 Motivation

Under-approximation in hybrid systems reachability analysis simplifies complex problems. This is because it focuses on a bounded subset of reachable states, thus avoiding the high complexity of exact or over-approximated reachability. Under-approximation aids scalability, making it feasible to analyze more complex systems by reducing the computational burden. Finding a bad state using the under-approximation, guarantees that the actual system can also reach them. It allows researchers to concentrate on specific, critical behaviors without the need to exhaustively analyze the entire state space. Examples of under-approximation are shown in Definition 3.3.3.

5.2.2 Example

In this section, we present an example to visualize how the under-approximation of the set minus operator works. We consider two H-Polytopes P and G in the 2D space. The example in Figure 5.13 shows two initial polytopes P and G in H-Representation. P consists of four constraints and G consists of three constraints.

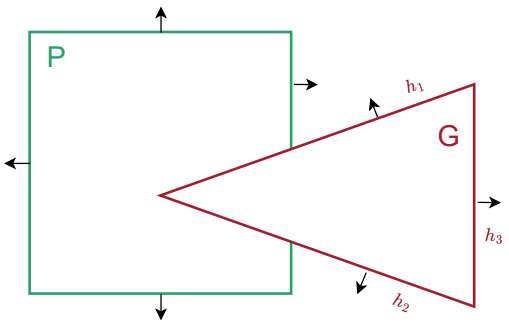
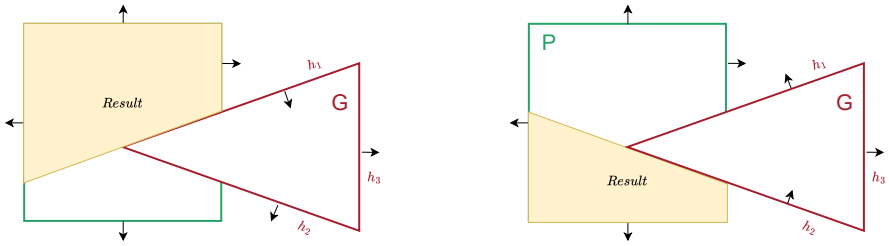


Figure 5.13: Initial polytopes P and G in the 2D space.

The idea is to convert each time one constraint of the polytope G and add it to the polytope P. Figure 5.14a shows a resulting polytope that is constructed with the constraints of the P and the inverted constraint h_1 of G. Analogously, the solution in Figure 5.14b is the resulting polytope that is created with the constraints of P and the inverted constraint h_2 of G. The example has three under-approximation solutions. However, we show only two, since the third solution results in an empty polytope. This happens when we invert the constraint h_3 of G. The aim of the set minus under-approximation is to return a single polytope. We choose the polytope with the largest volume. This is done, due to the fact that we analyze reachability and we want the biggest under-approximation. In this case, the polytope from Figure 5.14a is returned.



(a) Resulting polytope (yellow) of the set minus operator with an under-approximation after inverting constraint h_1.

(b) Resulting polytope (yellow) of the set minus operator with an under-approximation after inverting constraint h_2.

Figure 5.14: Possible under-approximation solutions.

5.2.3 Algorithm

In the following part of this chapter, we will introduce a simple pseudocode to compute the under-approximation of the set minus operator of two H-Polytopes.
Algorithm 10 visualizes as pseudocode the set minus under-approximation. The algorithm receives as input two H-Polytopes and returns an H-Polytope, which is an under-approximation of the set minus operation. We start by inverting a constraint of G and add it to a new polytope P'. The polytope P' contains the constraints of the original P and the current constraint from G that is inverted. We store the halfspaces of P' in the vectorPolytopes vector. This process is repeated for all constraints of G. After that, we iterate over the vector of polytopes that contain all possible solutions and compare the size of each polytope. We return the polytope with the largest size as the set minus under-approximation result.

Algorithm 10 setMinusUnder

Input: H-Polytope P, H-Polytope G
Output: H-Polytope($P \setminus G$)

```
1:  vectorPolytopes = []
2:  for constraint h ∈ G do
3:      invertedConstraint = h.invertConstraint()
4:      P' = P ∪ invertedConstraint
5:      vectorPolytopes.push_back(P')
6:  end for
7:  biggestPolytope = ∅
8:  biggestVolume = 0
9:  for polytope P' ∈ vectorPolytopes do
10:     currentVolume = P'.getEstimatedVolume()
11:     if currentVolume > biggestVolume then
12:         biggestPolytope = P'
13:         biggestVolume = currentVolume
14:     end if
15: end for
16: return biggestPolytope
```

What is left to show is how the size of a polytope can be evaluated. This thesis introduces a heuristic, which estimates the volume of a polytope. Algorithm 11 visualizes a pseudocode for estimating the volume of a polytope. The algorithm first calculates the minimum and maximum values for each dimension of the polytope. For that, we use linear programming to find the minimum and maximum values. After calculating the minimum and maximum values, the algorithm calculates an estimation box by multiplying the difference between the minimum and maximum values of each dimension. The algorithm then uses the Monte Carlo method to estimate the volume of the polytope [15]. It generates several random points in the estimation box and checks how many of them are inside the resulting polytope. The volume is then estimated by multiplying the size of the estimation box with the ratio of the number of points inside the polytope and the total number of points generated.

Algorithm 11 getEstimatedVolume

Input: H-Polytope $P = (\mathbf{a}_1,b_1), \cdots, (\mathbf{a}_n,b_n)$
Output: double volume

dimBounds = []
k = P.getDimension()
for int i = 0; i< k; i++ **do**
    // Calculate min and max value of each dimension

$$\min_{x_0,\cdots,x_k} \quad x_i \qquad \text{s.t.} \quad \forall_{1\leq i\leq n} \quad \mathbf{a}_i \cdot \begin{pmatrix} x_0 \\ \vdots \\ x_k \end{pmatrix} \leq b_i$$

$$\max_{x'_0,\cdots,x'_k} \quad x'_i \qquad \text{s.t.} \quad \forall_{1\leq i\leq n} \quad \mathbf{a}_i \cdot \begin{pmatrix} x'_0 \\ \vdots \\ x'_k \end{pmatrix} \leq b_i$$

    dimBounds.push_back((x_i, x'_i))
end for
estimationBox = $\prod_{i=0}^{k}(\text{dimBounds}[i].\max - \text{dimBounds}[i].\min)$

numberSamples = 1000
// Use Monte Carlo method to estimate the volume
numberPointsInside = useMonteCarlo(P, numberSamples, dimBounds)
volume = estimationBox $\cdot \frac{\text{numberPointsInside}}{\text{numberSamples}}$
return volume

A visualization of the volume estimation is seen in Figure 5.15. We show the estimation box (blue) of the polytope from Figure 5.14a. As shown in the figure, the difference of the maximum and minimum values regarding the first dimension equals 10. Similar to the second dimension the difference of the maximum and minimum value equals 8. These values are used to create the blue box. By multiplying the difference of the dimensions, we get the size of the estimation box, which is an overapproximation of the resulting polytope. The Monte Carlo method is used to estimate the volume of the actual polytope. The method generates several random points in

the estimation box and checks how many of them are inside the polytope.

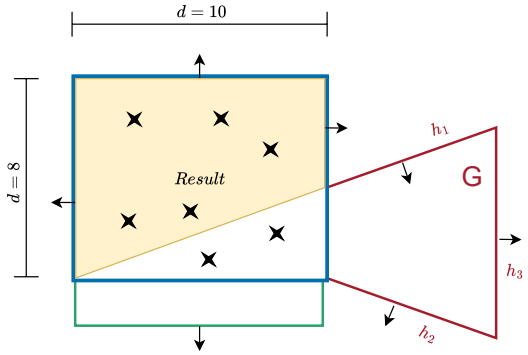


Figure 5.15: Monte Carlo method from Figure 5.14b.

Let 400 points be generated by the Monte Carlo method and let 300 of them be inside the yellow polytope. To estimate the volume of the polytope we multiply the volume of the box with the ratio of the number of points inside the polytope and the total number of points generated. In this case, the estimated volume of the polytope is:

$$\text{volume} = 10 \cdot 8 \cdot (\frac{300}{400}) = 60$$

This process is repeated for all polytopes that are created in Algorithm 10. The polytope with the largest estimated volume is returned as the result of the under-approximation set minus operator.

Chapter 6

Results

In this chapter, we present the results of the proposed algorithms. We present in Section 6.1 three different automata and plot their flowpipes. We show two cases, one without urgent and one with urgent transitions. This shows the correct function of the over-approximation algorithm. In Section 6.2, we show the results of the under-approximation algorithm. We illustrate a plot that shows the under-approximated convex set minus result. In the last Section 6.3, we compare the runtime of the proposed over-approximating algorithm with the already existing setMinus-Polytopes 2 algorithm.

6.1 Over-approximation Results

We begin by presenting three examples that were used to showcase the applicability of the above-presented algorithm. For all three examples, we introduce the hybrid automaton that is used. After that, we show the constructed flowpipes once without the jump being urgent and once with the jump being urgent. The blue segments in the flowpipes are the P segments as they evolve over time. The red boxes indicate the guard of the transition.

6.1.1 First Example

In Figure 6.1, we see the hybrid automaton that is used for the first example. It consists of two locations l_0 and l_1 with a jump between them, and it is a two-dimensional system with variables x and y. The initial value of the variable x is 0 and y is within the interval [1,2]. In location l_0, both variables evolve at the same rate, while in location l_1 they do not evolve at all. The transition from l_0 to l_1 is enabled when $x \in$ [1,1.5] and $y \in$ [2.5,3].

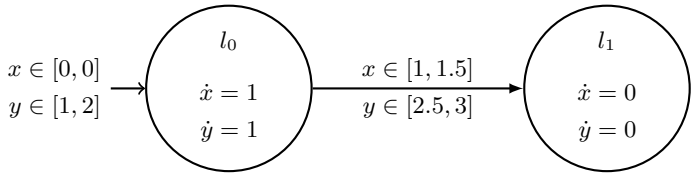
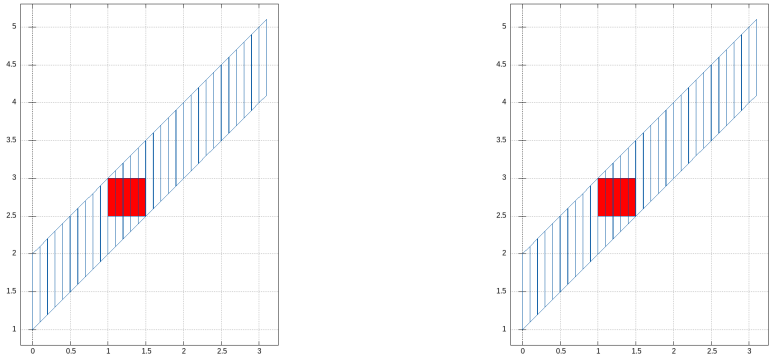


Figure 6.1: Example 1: Hybrid automaton.

Figure 6.2a shows the result of Algorithm 2 when the transition of the location l_0 is not marked as urgent. The reachable states evolve normally without the need for the computation of set minus operator since the jump is not forced to be taken. We see in both figures, Figure 6.2a and Figure 6.2b that the guard is the box, which is the cross product of the intervals $x \in$ [1,1.5] and $y \in$ [2.5,3]. In Figure 6.2b, we show the result of the algorithm when the transition of the location l_0 is marked as urgent. The reachable states are computed with the set minus operator since the jump is forced to be taken. In this scenario, we do not see a difference from Figure 6.2a. This is because the convex set minus of each polytope P that intersects with the guard is again the polytope P itself. As presented in Figure 5.2, the over-approximation can result in the initial polytope P.



(a) Example 1: Without urgent transition. (b) Example 1: With urgent transition.

Figure 6.2: Flowpipes of the first example.

6.1.2 Second Example

In Figure 6.3, the hybrid automaton that is used for the second example is shown. Again, the same two variables with the same initial values are used. In location l_0 the variable x evolves at a rate of 1, whereas the variable y evolves at a rate of 0. In location l_1 both variables evolve at a rate of 0. The transition from l_0 to l_1 is enabled when $x \in$ [2,2.5] and $y \in$ [1,1.5].

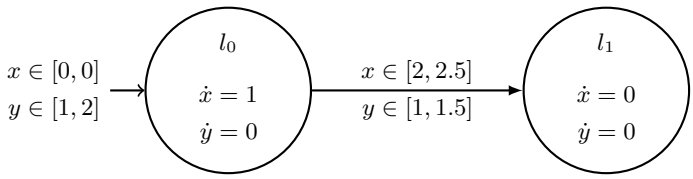
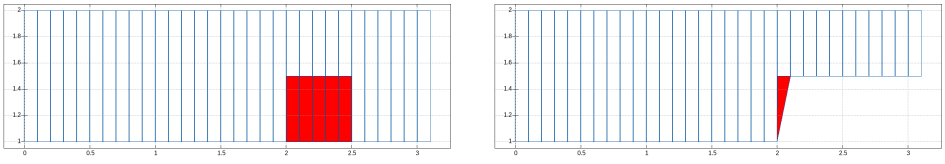


Figure 6.3: Example 2: Hybrid automaton.

In Figure 6.4a, we show the reachability analysis with time horizon 3 without urgent jumps. In Figure 6.4b, we see the difference that occurs when we mark the transition from location l_0 to l_1 as urgent. Using the introduced set minus crossing algorithm, we can see that the reachable states are computed differently. The guard is within the bounds $x \in$ [2,2.5] and $y \in$ [1,1.5]. After computing the set minus

of the first polytope that intersects with the guard, we get the resulting polytope which is shown in $x \in [2, 2.1]$ and $y \in [1,2]$. The crossing points of that polytope are (2,1), (2,2), (2.1,2), (2.1, 1.5) and thus the result. For the following segments, we see that the result is different from the first intersecting segment. This is because the crossing points for the next segment are (2.1,1.5),(2.1,2),(2.2,2),(2.2,1.5).



(a) Example 2: Without urgent transition. (b) Example 2: With urgent transition.

Figure 6.4: Flowpipes of the second example.

6.1.3 Third Example

For the last example, we use the hybrid automaton shown in Figure 6.5. We have the same two variables with the same initial values and evolve rates. The transition from l_0 to l_1 is enabled when $-x + y \leq -2.5$.

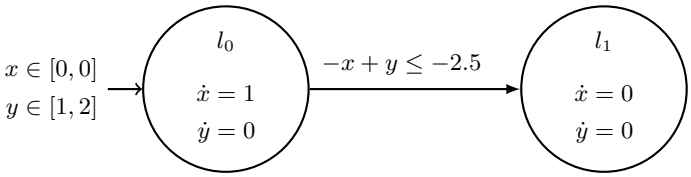
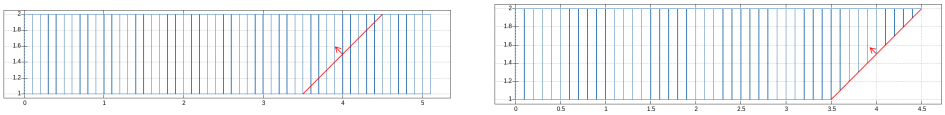


Figure 6.5: Example 3: Hybrid automaton.

This scenario differentiates from the other two since the guard is unbounded. For this case, we use Algorithm 9 where both initial polytopes are in H-Representation. In the first two scenarios, the representation type is in V-Representation.

In Figures 6.6a and 6.6b, the guard represents the inequality $-x + y \leq -2.5$. In Figure 6.4a, where the urgent transition is not enabled, we see no difference in the reachable states. On the other hand, in Figure 6.4b, we see that as soon as the unbounded guard is enabled, the reachable states are computed with the set minus operator and thus, we get a different result. Each intersection with the P polytope and the guard results in a smaller polytope.

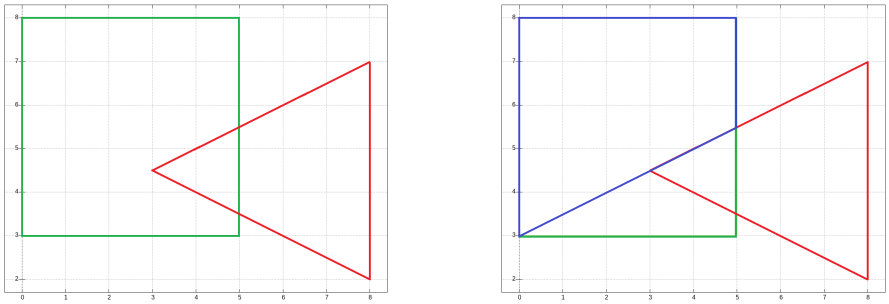


(a) Example 3: Without urgent transition. (b) Example 3: With urgent transition.

Figure 6.6: Flowpipes of the third example.

6.2 Under-approximation Results

In Figures 6.7a and 6.7b, we show the results of the under-approximation algorithm. We use the same example that was mentioned in the previous Section 5.2.2. Figure 6.7a shows the polytope P (green) and the guard G (red) in 2D space. The polytopes are in H-Representation. The result after applying the under-approximation algorithm is shown with the blue polytope in Figure 6.7b. This is the expected result since the blue polytope is the largest in terms of volume.



(a) Initial polytope P (green) and G (red) in the 2D space.

(b) Blue polytope indicates the result of the under-approximation.

Figure 6.7: Under-approximation of the polytope P without the guard G.

6.3 Runtime Comparison Results

For this section, additional experimental models have been used to compare the runtime of the proposed algorithm with over-approximation with the already existing setMinus-Polytopes algorithm.
In this context, we examine the runtime differences between the setMinus-Polytopes algorithm 2 and the algorithm proposed in this thesis. The goal is to highlight their respective capabilities and limitations in terms of runtime. For the runtime comparison, we use the same models for both of the algorithms and measure the average time it takes to compute the reachable states. One limitation of the setMinus-Polytopes

algorithm is that it only works with polytopes in H-Representation. For this reason, we use H-Representation for the comparison of the two algorithms. In the models, we use the benchmark parameters shown in Table 6.1.

Model	*#Var*	*#Loc*	*#Edge*	*#Urg*	Time horizon	Jump depth
vehicle_urgent	4	2	7	6	20	∞
moving_guard	2	10	9	9	20	[1,7]

Table 6.1: Parameters of the models used for the runtime comparison.

6.3.1 First Model

The first model that we used for benchmarking is an automaton proposed in [5]. The model is implemented as vehicle_urgent.model in the HyPro library. The variables x and y represent the position of the vehicle. The variable vx shows the velocity of the vehicle in the x direction. The vehicle starts at position $x = 0$ and $y \in [0, 5]$. As shown in Figure 6.8, there exist several urgent transitions represented by dashed lines. These transitions are enabled when the vehicle reaches certain positions. The vehicle is forced to brake for one second at these positions. After that, the vehicle is allowed to accelerate again.

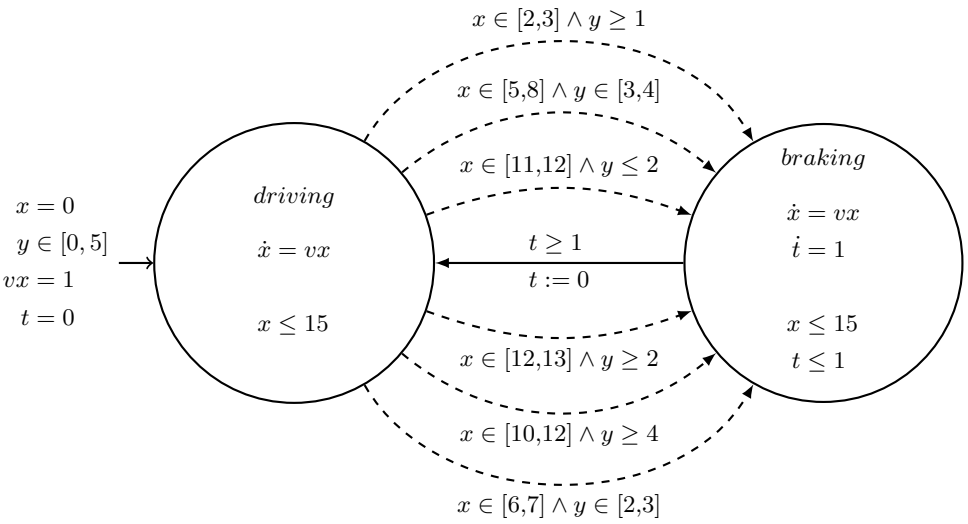


Figure 6.8: Vehicle model.

In Table 6.2 we see the runtime results of the two algorithms. We compute the reachability of the model for 40 iterations and measure the average runtime for the vehicle_urgent model. We measure the time in milliseconds and it is shown, that the setMinusCrossing algorithm is slightly slower than the setMinus-Polytopes algorithm. A possible reason for that is that the setMinus-Polytopes algorithm is optimized for the case where both polytopes are in H-Representation. The proposed algorithm as mentioned in Section 5.1.4, uses the setMinusCrossingH, when dealing with H-Polytopes. This function converts the polytope P into V-Representation. Due to this conversion, the runtime of the algorithm increases.

Model	vehicle_urgent	
Algorithm	setMinus-Polytopes	setMinusCrossing
Average runtime in ms	273.09	311.67

Table 6.2: Runtime comparison of setMinus-Polytopes and setMinusCrossing with 40 iterations for the vehicle model.

6.3.2 Second Model

The setMinus-Polytopes algorithm has a significant drawback due to its generation of numerous intermediate results during the set minus computation. Each setMinus-Polytopes operation results in a potentially huge set of polytopes, and applying the set minus operation multiple times can lead to exponential growth in the number of resulting polytopes, which can be computationally expensive. To highlight this weakness, we propose a model with multiple locations and guards that intersect with the polytopes. This simulates a scenario where the setMinus-Polytopes algorithm has to compute the set minus of a large number of polytopes. The model consists of 10 locations and two variables x,y. Initially, the variables are set to $x = 1$ and $y \in [1,20]$. While this model is a hand-made example, it simulates hybrid automata with a high number of guards. For simplicity reasons, we visualize in Figure 6.9 an example with three locations. The idea is to have a chain of locations, each connected by urgent jumps with an increasing bound as the model progresses. The first transition consists of the guard $2 \leq x$ and $2 \leq y$. After taking the jump, the new location only consists of a transition with the guard $4 \leq x$ and $4 \leq y$. This pattern continues until the last location, where the guard is $18 \leq x$ and $18 \leq y$. By introducing new constraints, the setMinus-Polytopes algorithm has to compute new polytopes and insert them into the set of possible states. This results in an exponential growth of the number of polytopes that the algorithm has to compute.

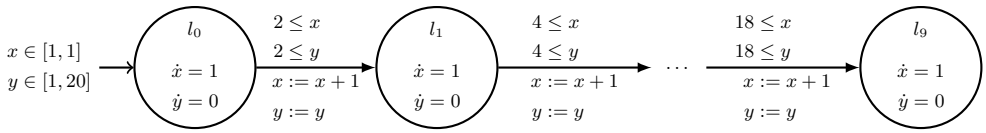
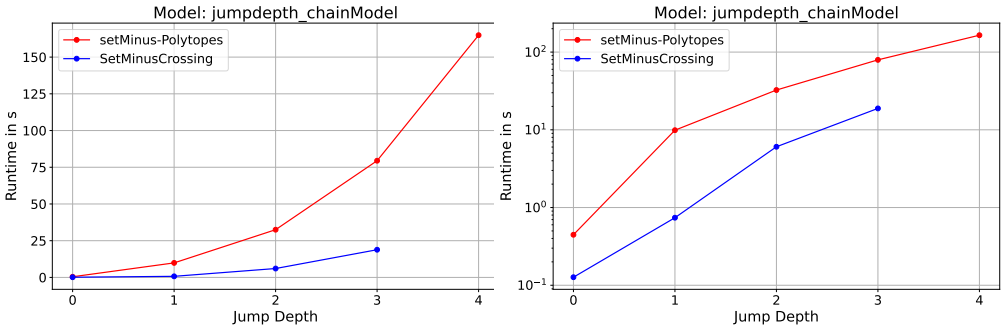


Figure 6.9: Model with a chain of locations.

In the following Figures 6.10a and 6.10b, we present the runtime results of the two algorithms. We show that as the number of jump depth increases, the runtime of the setMinus-Polytopes algorithm increases exponentially. The model from Figure 6.9 is used for this comparison. For a fixed jump depth, we used 5 iterations to evaluate the average runtime of the two algorithms. In addition, we set the time horizon to 20 and the time step to $\frac{1}{7}$ for both algorithms. The y-axis shows the runtime in seconds and the x-axis shows the number of jumps that the algorithms used. The data points represent the average runtime of the 5 iterations of each algorithm respectfully. We see that the setMinus-Polytopes algorithm increases exponentially with the number of jumps. While the plot does not show whether the setMinusCrossing algorithm has a linear or exponential growth, we can see that the runtime is significantly lower than the setMinus-Polytopes algorithm. Additionally, we could not compute the setMinusCrossing algorithm for a jump depth of greater than 3. This is due to current

library issues that need to be resolved in the future. Currently, an intermediate result of the flowpipe computation is intersected with the guard. After intersecting, a computed H-Polytope is checked for redundant constraints. This check is done by using an optimizer, which under certain values of the constraints fails to find a solution. This problem, however, is unrelated to the proposed algorithm itself.

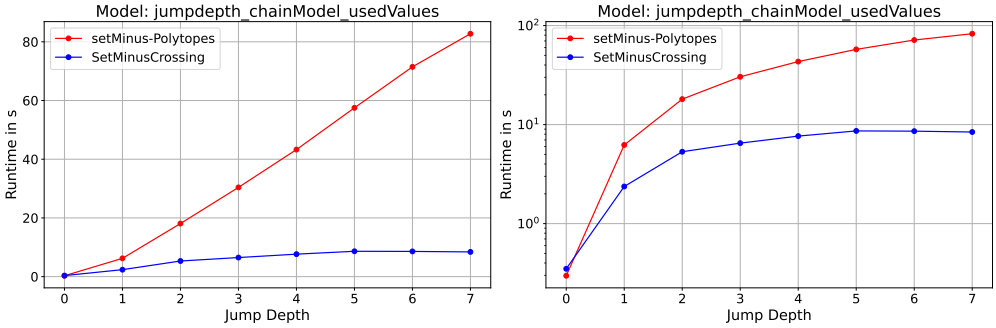


(a) Runtime comparison with average time of 5 iterations.

(b) Runtime comparison with average time of 5 iterations using log.

Figure 6.10: Runtime comparison of setMinus-Polytopes and setMinusCrossing with 5 iterations for the model from Figure 6.9 with time horizon = 20 and time step = $\frac{1}{7}$.

Previous results, shown in Figures 6.11a and 6.11b, are obtained by slightly changing the parameters of the model from Figure 6.9. While these results are currently not reproducible due to the mentioned library issues, they show that the runtime of the setMinusCrossing algorithm can be significantly lower than the setMinus-Polytopes algorithm.
All in all, the runtime comparison shows that the setMinusCrossing algorithm is a viable alternative to the setMinus-Polytopes algorithm.



(a) Runtime comparison with average time of 5 iterations.

(b) Runtime comparison with average time of 5 iterations using log.

Figure 6.11: Runtime comparison of setMinus-Polytopes and setMinusCrossing with 5 iterations for the model from Figure 6.9 with slightly different parameters.

Chapter 7

Conclusion

7.1 Summary

In this thesis, we extend the concept of hybrid automata by incorporating urgent transitions to analyze the reachability of urgent hybrid automata. Our primary objective was to enhance the HyPro library to handle urgency and to develop a robust algorithm capable of computing the set minus operator between two polytopes as an over-approximation.
To introduce the topic, we began with foundational examples and concepts, laying the groundwork for our subsequent developments. We present in Section 5.1 our algorithm, which computes an over-approximation of the set minus operator between two polytopes. This algorithm uses linear programming techniques to identify crossing points. We focus on two primary representations of state sets: the V-Representation and the H-Representation. Our algorithm is able to handle the set minus operation for both of these representations, as we demonstrate through detailed overviews and pseudocode illustrations.
In Section 5.1.3, we prove the correctness and efficiency of our algorithm through a series of lemmas.
Following this, we introduce in Section 5.2 a pseudocode for computing the set minus operation as an under-approximation between two polytopes. We present an illustrative example and a comprehensive explanation of its functionality.
In Section 6, we provided a series of examples and models to validate the correctness of the algorithms developed. For the over-approximation, three distinct examples are discussed, demonstrating the expected outcomes. Similarly, for the under-approximation, a single example is presented, confirming the accuracy of our approach.
Lastly, in Section 5.1.5, we compare the runtime performance of our proposed algorithm against the pre-existing setMinus-Polytopes algorithm in HyPro. Our findings indicate that, in the first example, our algorithm exhibits slightly slower performance due to the conversion of representations. However, in the second model, we highlighted a significant drawback of the setMinus-Polytopes algorithm: its inefficiency when dealing with many guards which can lead to reachability tree explosion. The setMinus-Polytopes algorithm, designed for polytopes in H-Representation, generates numerous intermediate results during the set minus computation. Each constraint conversion results in a set of polytopes, and applying the set minus operation to

each can lead to exponential growth in the number of resulting polytopes, which is computationally expensive.
In contrast, the algorithm proposed in this thesis can handle both polytopes in V-Representation and cases where the guard is in H-Representation. A key advantage of our algorithm is that it only produces one resulting polytope. This approach not only enhances computational efficiency but also simplifies the process of computing the set minus operation. Consequently, in scenarios involving many guards, our algorithm demonstrated superior performance, being much faster and more efficient. Important to note is, that our algorithm is an over-approximation whereas the setMinus-Polytopes algorithm computes an exact result.
In summary, this thesis contributes an algorithm for the over-approximation and under-approximation of the set minus operator between polytopes in the context of urgent hybrid automata. The developed methods not only extend the capabilities of the HyPro library but also offer significant improvements in computational efficiency in specific scenarios.

7.2 Future work

Due to time limitations, some improvements and extensions could be pursued in future work. These enhancements can further optimize the algorithm and extend the capabilities of the HyPro library for more complex representations and urgent hybrid automata.
Firstly, an important area for improvement is the runtime efficiency of our algorithm. As mentioned earlier, our algorithm can be slightly slower than the setMinus-Polytopes algorithm due to the conversion between polytope representations. We attempted to address this by introducing a new variable in the H-Polytope class to store the extreme points of the corresponding V-Polytope. These extreme points are created from all constraints of the H-Polytope. This approach aims to eliminate the initial conversion from H to V-Polytope, thus saving computational time. However, our tests indicated that this modification did not result in significant speed improvements, suggesting that further optimization is necessary, in order to be faster than setMinus-Polytopes algorithm. Exploring alternative implementations of the set minus operator for H-Representation that do not require conversion between polytope representations could lead to faster and more efficient algorithms. Developing an algorithm for the set minus operator to handle polytopes directly in their representations, without conversions, would avoid additional computational overhead.
Another promising direction is the integration of the set minus operator as an under-approximation within the urgent reachability analysis of HyPro. Although the primary focus of this thesis was on developing and validating the over-approximation algorithm, incorporating the under-approximation could provide a more comprehensive tool for reachability analysis in urgent hybrid automata. This step remains to be proved and tested.
Furthermore, future work could also include the development of an under-approximation algorithm for the set minus operation in the V-Representation. This would expand the utility of the developed methods and provide additional flexibility in handling different polytope representations within the urgent hybrid automata framework.
During the experimental phase, it was shown that the function getExtremePoints takes the most time during the set minus calculation. Given that, it would be benefi-

cial to investigate ways to optimize this function to improve the overall performance of the algorithm. This could involve exploring alternative methods for computing extreme points or implementing more efficient algorithms for this purpose.
By addressing these areas, future research can build upon the foundation laid by this thesis to create more robust and efficient tools for analyzing urgent hybrid automata. These advancements will not only enhance the performance of reachability analysis but also broaden the scope of applications and improve the overall effectiveness of the HyPro library.

Bibliography

[1] E. Ábrahám. Modeling and analysis of hybrid systems. *Lecture Notes*, 2012.

[2] M. Alcaraz-Mejia, A. Parres-Peredo, I. Piza-Davila, and L. Gutierrez-Preciado. Modelling and simulation process of extended Petri nets with PNML and MATLAB/Simulink. *International Journal of Simulation Modelling (IJSIMM)*, 22(2), 2023.

[3] R. Alur. An algorithmic approach to the specification verification of hybrid systems. In *Workshop on Theory of Hybrid Systems, Denmark*, 1992.

[4] 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.

[5] T. Ebert. Cegar approach for handling urgency in hybrid systems. 2021.

[6] 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 *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*, pages 379–395. Springer, 2011.

[7] P.-H. Ho. *Automatic analysis of hybrid systems*. Cornell University, 1995.

[8] A. Kim. Computing set difference for the reachability analysis of hybrid systems. In *Bachelor's thesis*. RWTH Aachen University, Unpublished thesis, 2021.

[9] S. Minopoli and G. Frehse. Non-convex invariants and urgency conditions on linear hybrid automata. In *Formal Modeling and Analysis of Timed Systems: 12th International Conference, FORMATS 2014, Florence, Italy, September 8-10, 2014. Proceedings 12*, pages 176–190. Springer, 2014.

[10] M. Otter, H. Elmqvist, and S. E. Mattsson. Hybrid modeling in modelica based on the synchronous data flow principle. In *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design (Cat. No. 99TH8404)*, pages 151–157. IEEE, 1999.

[11] P. Schrammel and B. Jeannet. From hybrid data-flow languages to hybrid automata: A complete translation. In *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, pages 167–176, 2012.

[12] S. Schupp. *State set representations and their usage in the reachability analysis of hybrid systems.* PhD thesis, Dissertation, RWTH Aachen University, 2019, 2019.

[13] O. Stursberg and B. H. Krogh. Efficient representation and computation of reachable sets for hybrid systems. In *Hybrid Systems: Computation and Control: 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3–5, 2003 Proceedings 6*, pages 482–497. Springer, 2003.

[14] S. Tripakis, C. Stergiou, C. Shaver, and E. A. Lee. A modular formal semantics for ptolemy. *Mathematical Structures in Computer Science*, 23(4):834–881, 2013.

[15] J. Von Neumann and S. Ulam. Monte Carlo method. *National Bureau of Standards Applied Mathematics Series*, 12(1951):36, 1951.

[16] B. Xue, P. N. Mosaad, M. Fränzle, M. Chen, Y. Li, and N. Zhan. Safe over- and under-approximation of reachable sets for delay differential equations. In *Formal Modeling and Analysis of Timed Systems: 15th International Conference, FORMATS 2017, Berlin, Germany, September 5–7, 2017, Proceedings 15*, pages 281–299. Springer, 2017.

[17] G. M. Ziegler. *Lectures on polytopes*, volume 152. Springer Science & Business Media, 2012.