The present work was submitted to the LuFG Theory of Hybrid Systems

BACHELOR OF SCIENCE THESIS

STAR SET REPRESENTATIONS IN THE REACHABILITY ANALYSIS OF HYBRID SYSTEMS

Dogu Tamgac

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

Additional Advisor:
Dr. Stefan Schupp

Aachen, 26.11.2021

Erklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und singemäße Zitate wurden als solche gekennzeichnet.

Dogu Tamgac
Aachen, den 26. November 2021

Abstract

The technical systems in our lifes is increasing by every year. Most of them needs to be checked for production. An electronic thermostat is one them. Checking if it will be working in all conditions is important. This thermostat could be used in factories and production lines and a thermostat which does not work in all conditions could damage the factories, business and most likely the people that use the product, which is made by using the thermostat. As the thermostat exhibits both continuous and discrete properties it can be modelled as a hybrid automata. Meaning that this thermostat can be verified for all conditions using a flowpipe-construction-based reachability analysis. However, representing the state-set of a hybrid system is not trivial. Choosing a state-set representation is always a trade-off between the speed and the precision of the computation. In the rest of this thesis we will look at the recently popularized state-set representation star sets. We will analyze the precision and speed of the star sets and also explain how to compute the operations applied on the star set.

Contents

Chapter 1

Introduction

A cyberphysical system is a computer system in which the behaviour and the mechanism is controlled by the computer. Cyberphysical systems are also used in many industries such as health, robotics and aerospace. As they effect our lives heavily, the safety of them should be checked before they are used. Cyberphysical systems are most faithfully modelled as hybrid systems, because a cyberphysical system exhibits both discrete and continuous properties. After modelling the system as hybrid automaton, one must represent the state space of variables. Then, through reachability analysis we can verify the safety properties of a hybrid system. However, in general the exact computation of reachable states is undecidable [ACH$^+$95]. Though, using some techniques provides us a good estimation. One them is overapproximating the reachable sets as flowpipes, which consists of several convex segments. This convex segments can be geometrically represented in many different ways: boxes, $\mathcal{H}$-polytopes, support functions and also as star sets, which is the main topic of this work.

In [BD17a], [BD17b], [DV16] a new technique to verify safety properties of linear systems has been introduced. One of the core ideas behind this technique is the state set representation known as star sets. In this thesis we will further elaborate this state set representation, as it exhibits some interesting properties. The star set representation will also be implemented in the library HyPro and we will check the efficiency of this representation in flowpipe-construction-based reachability analysis compared to other state set representations.

Chapter 2

Preliminaries

In this chapter some preliminaries and notation will be presented, which are necessary for the rest of this work. The notation, that we are going to use throughout the thesis, the definition of the hybrid automaton and basic concepts of the reachability analysis will be introduced. Then, in the following some of the other state set representations will be explained, so that we can acquire a better intuition about the reachability analysis and they will be also necessary while explaining the star set representation.

2.1 Notation

We are going to introduce the notation that will be used in the rest of thesis as we are going to use concepts from many mathematical topics. Let $\mathbb{R}$ the set of the real numbers and $\mathbb{N}$ the set of natural numbers excluding zero. Until stated otherwise the lowercase letters such as a,b,c,d,... will represent the vectors in $\mathbb{R}^{n\times 1}$ with $n \in \mathbb{R}$. The upper case letters such as A,B,C,D,... will represent matrices in $\mathbb{R}^{m\times n}$ where $m \in \mathbb{N}$ and $n \in \mathbb{N}$. The letter m represents the number of rows and the letter n represents the number of columns in the matrix. The letter I will be representing the identity matrix for the given dimension. We will also use the block matrices frequently. The block matrices will be introduced here with an example. Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$ be matrices with m,n,p,$q \in \mathbb{R}$. Let C be a matrix with:

$$C = \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix}$$

Then $C \in \mathbb{R}^{(p+m)\times(q+n)}$. Also here the number 0 denotes a matrix filled with zeros with the appropriate dimension.

2.2 Hybrid Automata

Before we get into the star set representation and methods of computation for cases such as intersection and affine transformation, the notion of hybrid systems will be introduced. A hybrid system, as it is also can be seen from the word "hybrid", is a system with both discrete and continuous behavior which unifies both notions in one system[SFÁ19]. One of the most popular examples in hybrid systems is the bouncing ball example [CSM+15]. As this is not a very complicated and easy to understand example, it will be used throughout this thesis for better understanding. The example consist of a ball which is dropped from a certain height h_0 and is in a free fall state. The horizontal movement of the ball will not be taken into account. The velocity of the ball and the vertical distance from the ground (height) with respect to time will be examined. As the ball begins to fall the distance from the ground h begins to decrease while the speed v of the ball begins to increase as it is affected from the gravitational pull of the earth. The acceleration of the ball is equal to $g = 9.81\,\frac{\mathrm{m}}{\mathrm{s}}$. As seen this is a continuous change of the position h and velocity v which is represents the continuous part of the hybrid system. The discrete change happens as the ball touches the ground. When the ball touches the ground the velocity is inverted and also dampened by some factor $0 < c < 1$ [SFÁ19]. This immediate change of the velocity represents the discrete behaviour in the hybrid system. After that it begins to increase in height and decrease in velocity until the velocity reaches the point $v = 0$. Then the process is repeated like it is explained before. To be able to define a hybrid system, one must first be able to represent it as a mathematical model. With the following definition of syntax, it will be possible to represent a hybrid system with an automaton. We will use the definition from [Ábr15], [ACH+95].

Definition 2.2.1: Hybrid Automata

A hybrid automaton is a tuple

$$H = (Loc, Var, Lab, Edge, Flow, Inv, Init)$$

which consists of the following parts:

- Loc is a finite set of locations(also called modes).
- Var a set with finitely many variables. a A valuation $v : Var \to \mathbb{R}$, where V is the set of all valuations.
- Lab, is the finite set of edge labels.
- $Edge$ is a set of finitely many transitions with $Edge \subseteq Loc \times Lab \times 2^{V^2} \times Loc$.

- $Flow$ is a function defined as $Flow : Loc \rightarrow (\mathbb{R}^+ \rightarrow V)$. This maps time-invariant functions to each location. When $f(t) \in Flow(l) \Rightarrow f(t + t') \in Flow(l)$ for all $t'\mathbb{R}^+$, then $f(t)$ is a time-invariant function.
- Inv is a function with $Inv : Loc \rightarrow V$.
- $Init$ is aset of initial states. A state is a tuple $(l,v) \in Loc \times V$. Initial sets are a subset of set of all states Σ. $Init \subseteq \Sigma$.

Inherently, a hybrid automata is a finite state machine. With a set of variables x_0,...,$x_{n-1} \in Var$ whose values change with continuous dynamics named the $flow$. $flow$ is modelled by time-invariant functions $f(x_0,..,x_{n-1})$ in each location. The $flow$ is usually denoted by ordinary differential equations (ODEs). With these ODEs the state variables evolve continuously according to a function with the form $\dot{x}_i = f(x_0,..,x_{n-1})$. While the variables are changing their values according to the $flow$, only one location can be active. Also each location have their invariants Inv, which are logical constraints over the variables in Var. These invariants must be satisfied while the location is active. As there can be many locations, there can also be transitions between different locations. Via these transitions, from the transition set $Edge$, the hybrid automaton **can** change its active location to another location itself included(loop). However, these transitions(jumps) are only available if certain constraints, called the guards, are satisfied. This discrete jump between locations can trigger reset function, which causes an immediate change in variable valuations. The reset function must satisfy the invariant of the target location.

The definition given before defines the syntax of the hybrid automata. However, it does not define its semantics. The time evolution of a variable according to the flow, satisfying a guard and location invariant are not covered with this definition of syntax. Because of that we are going to expand our definition of hybrid automata with operational semantics.

Definition 2.2.2: Hybrid Automata: Operational Semantics

The behaviour of a hybrid automata

$$H = (Loc, Var, Lab, Edge, Flow, Inv, Init)$$

is defined by the following rules:

- Discrete Rule or $Rule_{jump}$:

$$\frac{(l,a,\mu,l') \in Edge \qquad (v,v') \in \mu \qquad v' \in Inv(l')}{(l,v) \xrightarrow{e} (l',v')}$$

- Time Rule or $Rule_{Flow}$:

$$\frac{f \in Flow(l) \wedge f(0) = v \wedge f(t) = v' \qquad t \geq 0 \wedge \forall 0 \geq t' \geq t : f(t') \in Inv(l)}{(l,v) \xrightarrow{t} (l,v')}$$

The discrete rule is describes when a transition can be taken. A transition from location l to l' can only be taken when there is an edge between two locations and when v and v' is a valid pair of valuations. Meaning that the set $\{v|(v,v') \in \mu\}$ needs to satisfied in order to take the transition. This is called a guard. The reset is a mapping of each valuation v satisfying the guard to a set of possible valuations $\{v'|(v,v') \in \mu\}$ after taking the transition. Also it is further stated that valuation v' should also satisfy the invariant at the location l'.

The time rule handles the continuous changes in the hybrid automata. Meaning that where no transition is taken. Because of that the location is not changed. However, the valuation changes with respect to time t. We assume that f is an activity (ODE) and v,v' denote the start and end valuation according to f. The time rule states that all points at time t', which is between 0 and t, should also satisfy $f(t') \in Inv(l)$. From these rules we can define the reachability of a state. Taking a step with either rule is defined as an $execution$ step $\rightarrow$. A sequence of states $\sigma_0 \rightarrow \sigma_1 \rightarrow ...$ with taking either rule, while $\sigma_0 = (l_0,v_0) \in Init$ and $v_0 \in Inv(l_0)$, is defined as a run. If there exists a run that goes through a state then the state is defined as $reachable$. For simplicity we can write for a state σ', which is reachable in finitely many $execution$ steps, $\sigma \rightarrow^* \sigma'$.

Example: Bouncing Ball

Here an example with graphical representation of a hybrid automata will be introduced as it is more intuitive.

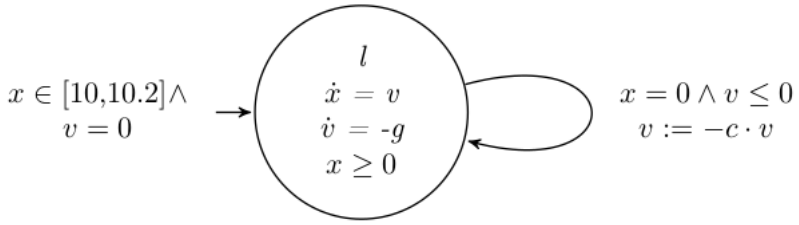


Figure 2.1: An example of a hybrid automata for bouncing ball system

As introduced before the bouncing ball system represents a ball falling from a certain height. In our case the value of the height is given as between $[10, 10.2]$. The reason that the height is given as a range of numbers is that ball is not a point but a physical object which takes up space. The initial velocity of the ball is 0. With time the ball starts to fall, the height and velocity of the ball changes with respect to the equations: $\dot{x} = v, \dot{v} = g$. When it touches the ground $x = 0$ it takes a discrete transition (here a loop). Because of that a reset function is applied to the velocity $v := -cv$. Notice that it is necessary to take the transition, when the ball touches the ground because the ball can not go through the ground $x \geq 0$. Normally if both the invariant of the active location and the rules for the discrete jump are satisfied, choosing an option is non-deterministic.

2.3 Reachability Analysis

In this section, after defining semantics and syntax of the hybrid automaton we will introduce its formal verification properties. Meaning that we are going to check, if it is possible to reach a given bad state with the initial inputs. As the bouncing ball system is already introduced, we are going to explain the intuition of reachability analysis through that example. For example is it possible for the ball to reach a velocity higher than $100\,\frac{\mathrm{m}}{\mathrm{s}}$? If we would like to verify that the ball does not reach a velocity higher than $100\,\frac{\mathrm{m}}{\mathrm{s}}$, we can use hybrid systems verification. While this is an easy question to solve using simple physics, most of the problems that are going to be examined will have higher dimensionality and therefore, they might not be so easy to solve using traditional approaches. The verification problem for hybrid systems could be formulated like:

Given:

- A hybrid automata H,

- Set of initial set $Init$,
- Set of bad states $Bad \subseteq \Sigma$,

Does the hybrid automata reach the bad states in time T? To be able to verify this we will define forward reachability, and introduce the forward reachability algorithm according to [Ábr15], [SFÁ19]. We are going to focus on iterative forward reachability problem (flowpipe-construction-based) in this work. This approach will be introduced more in detail after the general forward reachability algorithm is introduced.

Definition 2.3.1: Forward Reachability Problem

Let H be a hybrid automata, $Init$ be the initial set. The forward reachability is defined as computing the set of all states that are reachable from the initial set $Init$.

$$Reach(Init)^+ = \{\sigma' \in \Sigma | \exists \sigma \in Init : \sigma \rightarrow^* \sigma'\}$$

Definition 2.3.2: Forward Reachability Analysis

Let H be a hybrid automata, $Init$ be the initial set and $Bad \subseteq \Sigma$ be the set of bad states. The forward reachability analysis states that a hybrid system is safe for the given set of bad states Bad with:

$$Reach(Init)^+ \cap Bad = \emptyset$$

Before we get into the algorithm some intuition will be given on how this algorithm works. The general algorithm begins with the initial sets $Init$ to check and computes new states that is reachable from this the initial set $Init$. As long as a new state is reached, this process continues from the new computed states. When a new state can not be found, the process terminates with returning the reached states.

Algorithm 1: General Forward Reachability Algorithm

Input: Set of initial states $Init \subseteq \Sigma$
Output: Set of reachable states R

```
1 R := Init
2 R_new := R
3 while R_new ≠ ∅ do
4 |   R_new :=Reach(R_new)\R
5 |   R := R ∪ R_new
6 return R
```

2.3.1 Flowpipe Construction

The flowpipe construction algorithms use overapproximation to compute the reachable states [CK98]. It divides the computation into time segments. Let the time bound be $T \in \mathbb{R}$ and time step be $\delta = T/N$ where $N \in \mathbb{N}$ is the number of time steps. So that we have created N time segments in the form $[0,\delta]$,$[\delta,2\delta]$,...,$[(N-1)\delta,T]$. After dividing the computation to time segments, the reachable states in a time segment will be computed and overapproximation can be used. In particular, we will examine the affine hybrid automaton where the type of $Flow$ functions are restricted to the affine functions. Meaning that: The flow of every variable x_i must be in the form of $\dot{x_i} = f(x_0,...,x_{d-1})$ where $f(x_0,...,x_{d-1})$ is an affine function and d is the dimension. The definition of reachable states while using flowpipe construction is in the following:

Definition 2.3.3: Reachable States [Ábr15], [CK98]

Let $X_0 \subseteq \Sigma$ be the initial states. The set of reachable state $\mathcal{R}_{[}t,t']$, which are reachable from X_0 at the time interval $[t,t']$ is defined as:

$$\mathcal{R}_{[t,t']}(X_0) = \{x_s | \exists x_0 \in X_0 \wedge \exists t \leq s \leq t' \wedge x_s = x(s,x_0)\}$$

As seen from the definition that set of reachable states in time interval $[t,t']$ is the all valuations x_s, which are reachable from a initial state x_0. Analogously, the set of all reachable states is $\mathcal{R}_{[0,T]}(X_0)$, where T is the time bound.

We also can overapproximate $\mathcal{R}_{[0,T]}(X_0)$. The overapproximation can be done to speed up the computation and as the state set representations can not represent every shape [SFÁ19]. The over approximation will be done by overapproximating every segment $\mathcal{R}_{[(i-1)\delta,i\delta]}(X_0)$ $i \in [1,..,N]$. The union of all of the overapproximated segments gives the whole flowpipe approximation.

2.3.2 First Segment Computation

Given the initial set X_0, to compute the first segment following operations are needed: X_0 will transformed with the corresponding flow. As the flow of an affine is an affine vector field where every variable x_i follows the linear differential equation $\dot{x} = f(x_0,...,x_d - 1)$. The linear ODE can be approximated for the next valuation vector $x_{t,x_{init}}$ via the matrix exponentiation [Tse20], [CK98]:

$$x(t,x_{init}) = e^{a\delta}x_{init} = \sum_{k=0}^{\infty} \frac{(A\delta)^k}{k!}$$

With generalizing to the state sets:

$$X_\delta = e^{a\delta}X_0$$

The computation of convex hull of X_0 and X_δ excludes possible nonlinear trajectories. Because of that we will bloat X_δ with a box B using Minkowski sum. Then the first segment within time interval $[0,\delta]$ will be given as:

$$\hat{R}_{[0,\delta]} = conv(X_0(e^{a\delta}X_0 \oplus B))$$

In the given equation above $\hat{R}_{[0,\delta]}$ depicts the overapproximation of $R_{[0,\delta]}$, the symbol $\oplus$ means the Minkowski sum and the word $conv$ depicts the convex hull.

In the following a pseudo code algorithm will be given from the work [SFÁ19], which is used for the flowpipe construction based reachability analysis.

Algorithm 2: Bounded flowpipe-construction-based reachability analysis [SFÁ19]

Input: Hybrid automaton $\mathcal{H} = (Loc,Var,Lab,Flow,Inv,Edge,Init)$
Output: Overapproximated set of reachable states in $\mathcal{H}$

```
1  Q := Init
2  R := ∅
3  while Q ≠ ∅ do
4      while not timeBoundReached() do
5          Ω := Ω ∩ Inv(l)
6          R := Ω ∩ Inv(l)
7          if not jumpBoundReached() then
8              for (l,g,r,l') ∈ Edge do
9                  addElement(Q,r(Ω ∩ g) ∩ Inv(l'))
10         Ω := letTimePass(Ω)
11 return R
```

State set representations. States can be described by convex geometric objects. The choice of state set representation affects the the outcome of a given hybrid system. As we have mentioned before the state set representations are in a trade-off between precision and speed. Because of that all

of them come with their advantages and disadvantages. Star sets are also a state set representation and it is the main topic of this thesis. However, before starting to introduce the star sets, the definition of the polytopes will be given. Since they will be needed for star sets and they will be used frequently in this work.

2.3.3 Polytopes

As we are going to use the $\mathcal{H}$-polytope and $\mathcal{V}$-polytope many times in our explanation of the methods for the star sets, which will be used for the reachability analysis, the definition of them are given below.

Definition 2.3.4: $\mathcal{H}$-polytope and $\mathcal{V}$-polytope[SFÁ19]

A d-dimensional convex polytope $P_{\mathcal{H}}$ in $\mathcal{H}$-representation is a pair (N,c) with $N \in \mathbb{R}^{m \times d}$ and $c \in \mathbb{R}^m$, which defines a convex set:

$$P_{\mathcal{H}} = \bigcap_{i=0}^{m-1} h_i$$

as the intersection of finitely many half-spaces $\{h_0,...,h_m - 1\}$ with $h_i = \{x \in \mathbb{R}^d | n_{i,\_} * x \leq c_i\}$ ($n_{i,\_}$ refers to the i-th rows of N)
The same polytope can also be represented as the convex hull of a finite set $P_{\mathcal{V}} = \{v_0,...,v_{v_m-1}\}$ of vertices $v_i \in \mathbb{R}^d$(V-Representation):

$$P_{\mathcal{V}} = \{x | x = \sum_{i=0}^{m-1} \lambda_i * v_i \wedge \sum_{i=0}^{m-1} \lambda_i = 1 \wedge \lambda_i \in [0,1]\}$$

Chapter 3

Star Sets and Implementation

Although computing the reachable set of a hybrid automaton is possible by using other representations such as $\mathcal{H}$-Polytope or $\mathcal{V}$-Polytope approach, it is not as computationally efficient and not scalable as star sets in some aspects[TCD$^+$19]. The star sets are very efficient in affine mapping operations and it outperforms the polytope based representations because it is able to do both affine mapping and intersection with half-spaces operations efficiently in reachability analysis .
Before jumping directly to the formal definition of star set some intuition will be given in the following. A set S in $\mathbb{R}^n$ is called a star set if there exists a point c in S where all points x in S can be reached with a line segment which is also in S starting from c. Intuitively, this means that if we think set S as a region with boundaries, we can select a center point c and we can reach all points in S with a straight line without leaving the region S. As seen from the name star sets this allows us to define all convex shapes and also some non-convex shapes. The definition of a star set and its essential properties are given in the following.

Definition 3.0.1: Star Sets[BD17b]

A star set (A generalized star set or simply star) Θ is a tuple $\langle c, V, P\rangle$ where $c \in \mathbb{R}^n$ is called the center, $V = \{v_1, v_2, ..., v_m\}$ is a set of m vectors in $\mathbb{R}^n$ called the basis vectors, and $P : \mathbb{R}^m \to \{\top, \bot\}$ is a predicate.
The set of states represented by the star set is given as:
$[\![\Theta]\!] = \{x | \exists \bar{\alpha} = [\alpha_1, ..., \alpha_m]^T \; such\; that\; x = c + \sum_{i=1}^{m} \alpha_i v_i \; and \; P(\bar{\alpha}) = \top\}$

Notation and Restrictions

*In this work we will restrict the predicates to be a conjunction of **linear constraints**, $P(\alpha) \triangleq A\alpha \leq d$ where for p linear constraints, $A \in \mathbb{R}^{p\times m}$, α is the vector of m-variables i.e., $\alpha = [\alpha_1, ..., \alpha_m]^T$, and $d \in \mathbb{R}^{p\times 1}$. The linear constraints can also be interpreted as a $\mathcal{H}$-Polytope. While implementing and explaining the algorithms, we will use a $\mathcal{H}$-Polytope to represent the linear constraints to speed up the implementation and to avoid redundancy in the explanation.*
This definition of star sets is also more general than the existing work [BD17a], [DV16] where a star set was restricted of having no more than n basis vectors. When computing the input effects as a star set, this generalization is crucial. This important generalization will be used when computing the Minkowski sum and union of two star sets. Furthermore, in the following chapter many comparisons between $\mathcal{H}$-polytopes and star sets will be shown, as the star set could be interpreted as a $\mathcal{H}$-polytope which also has a center and a basis. Any set given as a conjunction of linear constraints in the standard basis, can be instantly translated to the star representation by taking the center as the origin, the n basis vectors as the standard orthonormal basis vectors and the original conjunction linear condition with each x_i replaced by α_i. Because of that it possible to interpret a $\mathcal{H}$-polytope as a star set with basis being the standard basis and center being the origin with respect to the dimension of the $\mathcal{H}$-polytope. So that it is reasonable to assume that the set of initial states Θ is given as a star set.
Also it will be assumed that the initial star set before all transformations is bounded and sometimes it is going to be referred to both the tuple Θ and the set of states $[\![\Theta]\!]$ as Θ. Also the letter V is going to be used for both generator matrix and set of basis vectors. The generator matrix is the matrix where the basis vectors are the columns. For example when V is referred as a set of standard basis vectors in $\mathbb{R}^2$:

$$V = \{\begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}\}$$

When V is referred as the generator matrix of the set of standard basis vectors in $\mathbb{R}^2$:

$$V = \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$$

An Example Star Set

For the star set $\Theta = \langle c, V, P\rangle$:

Take $V = \{\begin{pmatrix}2\\0\end{pmatrix}, \begin{pmatrix}0\\2\end{pmatrix}\}$ the scaled unit vectors and $c = (3,3)$

Consider $P(\bar{\alpha}) = A\bar{\alpha} \leq d$ where $A = \begin{pmatrix}1 & 0\\ -1 & 0\\ 0 & 1\\ 0 & -1\end{pmatrix}$ and $d = \begin{pmatrix}1\\1\\1\\1\end{pmatrix}$

The star set $\Theta = \langle c, V, P \rangle$ then defines the rectangular set:

$$[\![\Theta]\!] = \{(x,y) | 1 \leq x \leq 5 \wedge 1 \leq y \leq 5\}$$

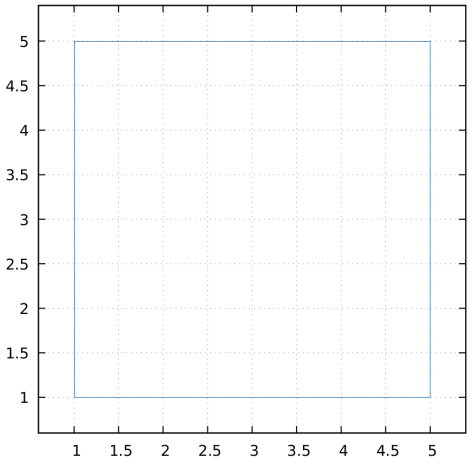


Figure 3.1: The example given corresponds to this figure given. Even if the predicates represent a box with side length 2 positioned at the origin. The center and basis of the star set affects the position and size of the star set so that the box has side length 4 and centered at $(3,3)$

Clarifications about algorithms and implementation
To write a cleaner and more understandable algorithm some basic methods will be introduced. To represent the properties such as center, predicates, etc. the following notation will be used:

- generator() returns the generator matrix V
- constraints() returns the constraint matrix A from the predicate P
- limits() returns d as a vector from the predicate P
- center() returns the center c coordinates as a vector

- predicates() returns a $\mathcal{H}$-polytope constructed from star sets constraints and limits

- starset(c,V,A,d) refers to a star set with center c, the generator V and predicates with $P(\bar{\alpha}) \triangleq A\bar{\alpha} \leq d$.

3.1 Affine Transformation

The affine transformation demonstrates the usefulness of star sets by computationally outperforming most of the other state set representations in this aspect. The affine transformation can be thought as a linear transformation followed by a translation.

To apply a linear transformation on the given star set, one must multiply the linear transformation matrix with the generator matrix and the center of the star set. After the multiplication the basis and center of the star set will change according to the linear transformation matrix. *The predicates do not need to change.* After that transformation the translation vector (i.e. offset vector) must be added to the center to achieve the desired translation. More formally:

Let the star Set $\Theta = \langle c,V,P \rangle$ with $V_n = \{v_1,v_2,...,v_n\}$ being a set of n vectors in $\mathbb{R}^n$ called the basis vectors, $c,x \in \mathbb{R}^n$ and $g \in \mathbb{R}$, $n \in \mathbb{N}$ with P being arbitrary linear constraints. An affine mapping of the star set Θ with the affine mapping matrix W and offset vector b is defined by another star set with the following characteristics.[TCD+19]

$$\bar{\Theta} = \langle \bar{c},\bar{V},P \rangle, \bar{c} = Wc + b, \bar{V} = \{Wv_1,Wv_2,...,Wv_n\}$$
$$[\![\bar{\Theta}]\!] = \{Wx + b \mid x \in \Theta\}$$

Algorithm 3: Affine Transformation

Input: Star set Θ, Linear Transformation Matrix W, Offset vector b
Output: Star set

1 Θ.generator() $\leftarrow W \cdot \Theta$.generator();
2 Θ.center()$\leftarrow W \cdot \Theta$.center() +$b$;
3 return Θ;

As the affine transformation only consists of two matrix multiplications and

one addition of vectors, it is more computationally scalable and more efficient than $\mathcal{H}$-polytopes while having linear constraints like the $\mathcal{H}$-polytopes.

An Example Affine Transformation on Star set

Let the star set $\Theta = \langle c, V, P \rangle$ *same as the first example given in the section before with:*

$V = \{\begin{pmatrix}2\\0\end{pmatrix}, \begin{pmatrix}0\\2\end{pmatrix}\}$, $c = (3,3)$, $P(\bar{\alpha}) = A\bar{\alpha} \leq d$ where $A = \begin{pmatrix}1 & 0\\ -1 & 0\\ 0 & 1\\ 0 & -1\end{pmatrix}$ and $d = \begin{pmatrix}1\\1\\1\\1\end{pmatrix}$ *The star set* $\Theta = \langle c, V, P \rangle$ *then defines the rectangular set:*

$$[\![\Theta]\!] = \{(x,y) | 1 \leq x \leq 5 \wedge 1 \leq y \leq 5\}$$

Also let the linear transformation matrix $W = \begin{pmatrix}cos(45°) & -sin(45°)\\ sin(45°) & cos(45°)\end{pmatrix}$ and the offset vector $b = \begin{pmatrix}1\\0\end{pmatrix}$. Let $\Theta' = L\Theta + b$. The resulting Θ' is defined as:

$$\Theta' = \langle Wc + b, WV, P \rangle$$

3.2 Intersection with Half-space

In this part it going to be explained how to compute and calculate the intersection between a star set and a half-space. The traditional approach of intersection between a half-space h and a $\mathcal{H}$-polytope A_H is only adding the half-space h as a constraint to A_H, following the definition of the $\mathcal{H}$-polytope, which is defined as the conjunction of half-spaces[SFÁ19]. However, as the star set also has a basis and a center unlike an $\mathcal{H}$-polytope this approach is not possible without transforming the half-space. The half-space will be rewritten using the basis and center of the star set. After this transformation it will be possible to just use the traditional approach of intersection between linear constraints and half-space. In the following a proposition [TCD$^+$19] about how to compute the intersection between a star set and a half-space will be introduced. Then the proof of the proposition will be given after.

Proposition 3.2.1: Star Set and Half-space Intersection

Intersection of a half-space $H \triangleq \{x|h^T x \leq g\}$ and a star $\Theta \triangleq \langle c,V,P\rangle$:

$$\bar{\Theta} = \Theta \cap H = \langle \bar{c}, \bar{V}, \bar{P}\rangle, \bar{c} = c, \bar{V} = V, \bar{P} = P \wedge P'$$
$$P'(\alpha) \triangleq (n^T V_n)\alpha \leq g - h^T c, V_n = \{v_1,v_2,...,v_n\}$$

Proof of the proposition

Let half-space H be defined as $H \triangleq \{x|n^T x \leq g\}$ and the star set $\Theta = \langle c,V,P\rangle$ with $V_m = \{v_1,v_2,...,v_m\}$ being a set of m vectors in $\mathbb{R}^n$ called the basis vectors, $c,x,h \in \mathbb{R}^n$, $g \in \mathbb{R}$ and $n \in \mathbb{N}$. As seen from the proposition above we are trying add a constraint to our predicate P to represent the intersection between the star set and the half-space. To achieve that, the equation $h^T x \leq g$ will be rewritten using the basis and the center of the star set. x from the half-space can be written as $x = c + \sum_{i=1}^{n} \alpha_i v_i$ with α_i being the coefficients, c being the center of the star set and v_i being the basis vectors of the star set:

$$h^T x \leq g$$
$$h^T(c + \textstyle\sum_{i=1}^{n} \alpha_i v_i) \leq g$$
$$h^T c + h^T(\textstyle\sum_{i=1}^{n} \alpha_i v_i) \leq g$$
$$h^T(\textstyle\sum_{i=1}^{n} \alpha_i v_i) \leq g - h^T c$$

For convenience we are going to write $\sum_{i=1}^{n} \alpha_i v_i$ as $V\alpha$ with $V \in \mathbb{R}^{n\times n}$ being the generator matrix, where the basis vectors are the columns and α being the coefficient vector. To show that $\sum_{i=1}^{n} \alpha_i v_i$ is equal to $V\alpha$:

$$V\alpha = \begin{bmatrix} | & | & & | \\ v_1 & v_2 & \dots & v_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} \alpha_1 v_{1,1} + \alpha_2 v_{2,1} + \ldots + \alpha_n v_{n,1} \\ \alpha_1 v_{1,2} + \alpha_2 v_{2,2} + \ldots + \alpha_n v_{n,2} \\ \vdots \\ \alpha_1 v_{1,n} + \alpha_2 v_{2,n} + \ldots + \alpha_n v_{n,n} \end{bmatrix}$$

$$= \alpha_1 v_1 + \alpha_2 v_2 + ... + \alpha_n v_n = \sum_{i=1}^{n} \alpha_i v_i$$

So that: $\sum_{i=1}^{n} \alpha_i v_i = V\alpha$ and because of the associativity property of matrices in multiplication it can be written as:

$$(h^T V)\alpha \leq g - h^T c$$

With this proposition we are adding the constraint $(h^T V)$ to the end of the rows of our constraint matrix A and $(g - h^T c)$ to the end of d. The half-space intersection adds only one constraint to the our star set.

We can see that, a star set does not change its predicate over affine mapping operations, and it preserves the center and basis vectors in the intersection with a half-space.

Example
Let half-space H be defined as:

$$H \triangleq \{x | h^T x \leq g\} \text{ with } x \in \mathbb{R}^2, \; h = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \text{ and } g = 2$$

For the star set take:

$$V = \begin{pmatrix} 0{,}5 & 0 \\ 0 & 0{,}5 \end{pmatrix} \text{ be the generator matrix and } c = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

$$\text{Consider } d = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \text{ and } P(\bar{\alpha}) = A\bar{\alpha} \leq d \text{ where } A = \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}$$

The star set: $\Theta = \langle c, V, P \rangle$

Intersection of Star Set and Half-space:
Using the proposition only some constraints and limits are going to be added our predicate P. The new predicate is given as:

$$\text{Consider from the proposition } (h^T \cdot V) = \begin{pmatrix} 0{,}5 & 0{,}5 \end{pmatrix} \text{ and } g - h^T \cdot c = -4$$

$$\Rightarrow \bar{P}(\bar{\alpha}) = \bar{A}\bar{\alpha} \leq \bar{d} \text{ where } \bar{A} = \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \\ 0{,}5 & 0{,}5 \end{pmatrix} \text{ and } \bar{d} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ -4 \end{pmatrix}$$

The new star set becomes: $\Theta = \langle c, V, \bar{P} \rangle$

Implementation

Algorithm 4: Half-space Intersection

Input: Star set Θ, half-space H

Output: Star set

// $x = g - h^T c$

1 Number x $\leftarrow$ H.offset()-(H.normal().transpose())$\cdot$ (Θ.center())

2 y$\leftarrow$(H.normal().transpose()$\cdot V$) // $y = h^T * V$

3 A'$\leftarrow \begin{pmatrix} A \\ y \end{pmatrix}$

4 d'$\leftarrow \begin{pmatrix} d \\ n \end{pmatrix}$

5 return starset(c,V,A',d')

3.3 Conversions

In this part conversion to other set representations will be introduced. It will be assumed that the star set has before all transformations a standard basis and is also bounded. Also as the affine transformation is in already implemented in HyPro for other state set representations, we are going to use it in our conversion. The details for affine transformation on $\mathcal{H}$-polyopes can be found in the preliminaries section.

It will be also assumed that the basis vectors of the star set are not all zero vectors, meaning that at least one entry in one of the basis vectors has a value other than zero. If this assumption does not hold, it means that the center of the star set is the only point that the star set contains. Because of that this conversion, which will be introduced in the following, will not be necessary as it is a more expensive operation than just creating a set representation with the only one vertex, which is equal to the center of the star set.

From the definition of $\mathcal{H}$-polytope it is clear that the star set that has linear constraints only has a generator matrix and a center different than a $\mathcal{H}$-polytope. The idea behind converting the star set to other representations will be interpreting the basis and center, as a storage for all affine and linear transformations applied before. The conversions will be explained through the conversion to a $\mathcal{H}$-polytope because a star set is essentially a $\mathcal{H}$-polytope with a basis and center. If a star set could be converted to a $\mathcal{H}$-polytope then it can also be converted to other representations as it is explained in [SFÁ19].

The conversion to a $\mathcal{H}$-polytope is done as follows: Let $L_i \in \mathbb{R}^{n\times n}, \forall i \in \mathbb{N}$ be the linear transformation matrices, $b_i \in \mathbb{R}^n, \forall i \in \mathbb{N}$ the offset vectors and $\Theta = \langle c, V, P\rangle$ be a star set with $m \in \mathbb{R}$ standard basis vectors where the dimension of the basis vectors is $n \in \mathbb{R}$ and the center $c \in \mathbb{R}^n$ is the origin. When an affine transformation is applied on the star set the generator matrix is changed to $V' = L_i * V$ and the center to $c' = L_i * c + b_i$. Meaning that we can create a new $\mathcal{H}$-polytope with the same predicate and then an affine transformation can be used on the $\mathcal{H}$-polytope to obtain a $\mathcal{H}$-polytope which covers the same area as the star set. Because all of the linear transformations are used on the standard basis(generator matrix), the generator matrix can be used as the only linear transformation matrix which needs to be used on the $\mathcal{H}$-polytope. The center c' will also be used as the offset vector. We can generalize this approach where more than one affine transformation was applied on the star set. Meaning that our star set becomes $\Theta' = \langle c'', V'', P\rangle$ after multiple affine transformations where $V'' = L_n \cdot L_{n-1} \cdot ... \cdot L_1 \cdot V$ and $c'' = L_n \cdot (L_{n-1} \cdot ... \cdot (L_1 \cdot c + b_1) + ... + b_{n-1}) + b_n$ for arbitrary $L_i \in \mathbb{R}^{n\times n}, b_i \in \mathbb{R}^n, \forall i \in \mathbb{N}$. From the affine transformed star set a $\mathcal{H}$-polytope can be constructed with the same predicate P and then an affine transformation can be applied to the $\mathcal{H}$-polytope with the affine transformation matrix being V'' and offset vector being c'' on the $\mathcal{H}$-polytope.

An important thing to recognize is that, we use affine transformation even if only linear transformations are used on the star sets. Because if the star set is created with a center being a point other than the origin, then it means that we already have an offset vector.

It also needs to mentioned that while converting to other representations such as boxes, it is more efficient to construct the box from the predicates of the star set and then apply the affine transformation as explained before. The reason for this better efficiency is that constructing an $\mathcal{H}$-polytope from the predicates then applying affine transformation and then converting to box results in two transformations. Also if the basis of the star set is not invertable, the constructed $\mathcal{H}$-polytope must be converted to a $\mathcal{V}$-polytope to apply the affine transformation. The number of conversions to convert to the box-representation could result in 3 conversions, if this is not taken into consideration. Also the affine transformation for boxes is more efficient than the one for $\mathcal{H}$-polytope.[SFÁ19]

Implementation

Algorithm 5: Conversion to $\mathcal{H}$-polytope

Input: Star set Θ

Output: Polytope representation of the star set Θ

```
// Creating a H-polytope from the star set predicates
```

1 $\mathcal{H}$-polytope poly $\leftarrow$ $\mathcal{H}$-polytope(Θ.constraints(),Θ.limits())

```
// Affine transformation of H-polytope with the basis of star
   set
```

2 return poly.affineTransformation(Θ.generator(),Θ.center())

3.3.1 From $\mathcal{H}$-polytope

As seen above the conversion to $\mathcal{H}$-polytope from a star set is already handled and in this part it will be explained how a $\mathcal{H}$-polytope can be converted into a star set. A $\mathcal{H}$-polytope can be thought as a star set where the basis the standard basis and center is the origin. It means that it is only needed to create a star set with the same predicates and assign the center as origin and the basis as the standard basis with respect to its dimension. So that the conversion is straightforward.

Implementation

Algorithm 6: From $\mathcal{H}$-polytope to star set

Input: $\mathcal{H}$-polytope H

Output: Star set Θ

```
// Create vector filled with zeros, which has the same
   dimension as H
```

1 vector c$\leftarrow$Zero(H.dimension)

```
// Use standard basis for the star set basis
```

2 matrix V$\leftarrow$ Standard basis with same dimension as $\mathcal{H}$-polytope H

3 return starset(c,H.constraints(),H.limits(),V)

3.3.2 Contains Point

Checking if a point lies in the star set depends on the basis of the star set. A naive idea would be to check if the coordinates of a point satisfy the predicates of star set. However, the predicates are dependent on the basis of the star set and only checking if the constraints would return the wrong result. Another naive idea would also be converting the star set to a $\mathcal{H}$-polytope and then checking if the point lies in the $\mathcal{H}$-polytope. As this method involves conversion it is not really efficient. However, if the an

affine transformation would be used on the point with linear transformation matrix being the generator and offset vector being the center, then it can be checked directly if the point lies in the star set. The affine transformation for points are explained in the preliminaries section. As mentioned many times the predicates will be interpreted as a $\mathcal{H}$-polytope. After the application of the affine transformation on the point as explained above, it will be checked if the point satisfies the predicates($\mathcal{H}$-polytope) of the star set.

Algorithm 7: Contains point

Input: Star set Θ, Point p
Output: Boolean
1 vector p'=p.affineTransformation(Θ.generator(),Θ.center())
2 return Θ.predicates().contains(p')

3.3.3 Check for Emptiness

So far we have mentioned that the predicate of a star set could be seen as a $\mathcal{H}$-polytope. If the basis is constructed from only empty vectors (meaning only containing the number 0), it will be directly assumed that the star set consists of only one point, the center of the star set. Other than the case explained here, where the basis vectors are empty, the basis and center of a star set has no effect on the emptiness of the star set. The center of the star set shows where the star set lies and therefore it has no effect on the emptiness. When it comes to the basis it has no effect on the emptiness either other than the edge case which is already mentioned. Because of that the basis and center will not be taken into consideration, while checking the emptiness of the star set. Only the satisfiability of the linear constraints needs to be checked to determine if star set is empty. Testing whether predicate is satisfiable can be done via linear programming (LP). More formally:
Let the star set be $\Theta = \langle c, V, P \rangle$ where $P(\alpha) \triangleq A\alpha \leq d$ and $A \in \mathbb{R}^{p \times m}$, α is the vector of m-variables i.e., $\alpha = [\alpha_1, ..., \alpha_m]^T$, and $d \in \mathbb{R}^{p \times 1}$. Checking whether there exists an alpha such that $\{A\alpha \leq d | \alpha \in \mathbb{R}^p\} = \emptyset$ holds can be done using an LP- or SMT-solver like GLPK [Mak08], SMT-RAT [CKJ+15], SOPLEX [Wun96] or Z3 [DMB08].[SFÁ19]

3.3.4 Vertices

In this part how to find the vertices of a star set will be explained. A naive idea would be to convert the star set to an $\mathcal{H}$-polytope and then finding the vertices by conversion to an $\mathcal{V}$-polytope. The problem with this

approach is that by affine transformation while converting to a $\mathcal{H}$-polytope, a conversion to $\mathcal{V}$-polytope could happen if the basis is not invertable. As we converted to the $\mathcal{V}$-polytope representation it will again be converted to $\mathcal{H}$-polytope in HyPro. After that to find out the vertices it will be converted again to $\mathcal{V}$-polytope. This means that 3 conversions could take place if it is done naively.
The more efficient way to find out the vertices of a star set is by converting the star set to a $\mathcal{V}$-polytope. First, the predicate will be converted to a $\mathcal{V}$-polytope and then affine transformation will be used by taking basis as the linear transformation matrix and center as the offset vector. Because the $\mathcal{V}$-representation is based on the vertices of the polytope the vertices can be found easily.

3.4 Intersection of two Starsets

Let $\Theta_1 = \langle c_1, V_1, P_1 \rangle$ and $\Theta_2 = \langle c_2, V_2, P_2 \rangle$ be appropriate star sets which have the same dimension and are defined according the definition that is given at the beginning. A naive idea to compute their intersection would be to take their predicates and join them to construct a new star set. Meaning that a new star set is constructed with $\Theta' = \langle c_1, V_1, P_1 \wedge P_2 \rangle$. There are at least two problems that arises from this idea.
First the basis of both star sets can be different. Meaning that the predicates are not dependent on the same basis as both star sets can have a different basis. Even if the constraints would be the same, if the basis of two star sets are different they cover a different area. The second one is that star sets can have different centers. Because of that the easy and reliable choice would be to convert both star sets in to $\mathcal{H}$-polytopes, as proposed before and then compute the intersection. After that the intersection can be converted to back to a star set.

However, if the star sets share the same basis and center, there would be no need to convert them to $\mathcal{H}$-polytopes. We can just use the intersection of the linear constraints from both star sets as our linear constraints for our new star set which represents the intersection of the given two star sets.

3.5 Minkowski Sum

Before talking about Minkowski sum with star sets, some intuition and the general definition of Minkowski sum will be given. Then computation of

Minkowski sum for star sets will be introduced.

Definition 3.5.1: Minkowski sum[SFÁ19]

The Minkowski sum $A \oplus B$ of two sets A,$B \subseteq \mathbb{R}^d$ is defined as the set

$$\{a + b \mid a \in A \wedge b \in B\}$$

and represent the set-theoretic equivalent to addition.

As seen from the definition convex sets are closed under the Minkowski sum. However, computation of Minkowski sum is normally exponential for $\mathcal{H}$-polytopes. Repeated computation of the Minkowski sum has a strong influence on the complexity of the resulting output.[SFÁ19] In the following a definition of Minkowski sum with star sets will be given.

Definition 3.5.2: Minkowski sum with star sets [BD17b]

Given two stars $\Theta = \langle c, V, P\rangle$ with m basis vectors and $\Theta' = \langle c', V', P'\rangle$ with m' basis vectors their Minkowski is a new star $\bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{P}\rangle$ with $m + m'$ basis vectors and
(i) $\bar{c} = c + c'$,
(ii) $\bar{V}$ is the list of $m+m'$ vectors produces by joining the list of basis vectors of Θ and Θ',
(iii) $\bar{P}(\bar{\alpha}) = P(\alpha_m) \wedge P'(\alpha_{m'})$ Here $\alpha_m \in \mathbb{R}^m$ denotes variables in Θ, $\alpha_{m'} \in \mathbb{R}^{m'}$ denotes variables for Θ' and $\bar{\alpha} \in \mathbb{R}^{m'+m}$ denotes variables for $\bar{\Theta}$(with appropriate variable renaming)

It is clear from the definition that both number of basis vectors and the number of constraints in the star set grow with each Minkowski sum operation. In an LP formulation of these constraints, this would mean that both the number of columns and the number rows grows at each step in the algorithm. At first glance this may seem like an inefficient way to compute the Minkowski sum of star sets. However, this in increase in size of the constraint matrix is not as bad as it first appears. Then the number of non-zero entries grows at each step. Meaning that, LP solvers that use a sparse matrix representation can compute the Minkowski sum of star sets efficiently [BD17b]. The algorithm of how to compute Minkowski sum of two star sets is given below .

Algorithm 8: Minkowski sum

Input: Star set Θ_1, star set Θ_2
Output: Star set Θ'

```
// Setting new center (i)
```
1 vector c'$\leftarrow$ Θ_1.center()+Θ_2.center()
```
// Setting the generator matrix (ii)
```
2 matrix V'$\leftarrow$ $\begin{pmatrix}\Theta_1.\text{generator}() & \Theta_2.\text{generator}()\end{pmatrix}$
```
// 0 represents a matrix filled with zeros with appropriate
   dimension
// setting new constraints matrix (iii)
```
3 matrix A'$\leftarrow$ $\begin{pmatrix}\Theta_1.\text{constraints}() & 0 \\ 0 & \Theta_2.\text{constraints}()\end{pmatrix}$
```
// Setting new limits vector (iii)
```
4 vector d'$\leftarrow$ $\begin{pmatrix}\Theta_1.\text{limits}() \\ \Theta_2.\text{limits}()\end{pmatrix}$
```
// Create a new star set with the new variables
```
5 return starset(c',V',A',d')

One of the things that needs to be mentioned is that, if the dimension of the problem is not too big the star set could be always converted to a $\mathcal{H}$-polytope to compute the Minkowski sum. Empirically speaking in problems with in lower-dimensional state spaces such as bouncing ball (Ball) test, using conversion method to compute Minkowski sum outperformed the method that is mentioned above.

3.5.1 Satisfies Half-space

In this method it will be shown how to find if a star set lies in a half-space. Meaning that all coordinates that satisfy the predicate of a star set should lie in the half-space. More formally:
Let half-space H be defined as $H \triangleq \{x|n^T x \leq g\}$ and the star Set $\Theta \triangleq \langle c,V,P\rangle$ with $V_n = \{v_1,v_2,...,v_n\}$ being a set of n vectors in $\mathbb{R}^n$ called the basis vectors, $c,x,n \in \mathbb{R}^n$ and $g \in \mathbb{R}$. If a star set satisfies the half-space H then:

$$\Theta \cap H = \Theta$$

Meaning that:

$$\forall x \in [\![\Theta]\!] \Rightarrow x \in H$$

To check if the star set satisfies the half-space, the intersection with half-space method, which is introduced before, will be used. First the intersection with star set will be computed and then it will be checked, if the star set still covers the same area. As we have seen before the intersection with half-space method does not change the basis and the center of the star set. If we think that the linear constraints as a $\mathcal{H}$-polytope and the basis, center as the affine transformation variables, it is clear that we only need check if the linear constraints($\mathcal{H}$-polytope) of the star set satisfies the half-space. Because of this phenomenon, in implementation a $\mathcal{H}$-polytope will be constructed from the linear constraints. Then the transformed half-space will be computed. After that the program will check if the $\mathcal{H}$-polytope satisfies the half-space. How the satisfaction of half-space is computed for $\mathcal{H}$-polytopes is introduced in the preliminary section.

3.6 Union

In this section we are going to present, how to compute the union of two star sets. Also the while explaining union method it must be mentioned that the resulting star set will be the convex hull of the star sets where it is the smallest possible closure of the union of the star sets. The union is used less frequently during reachability analysis than other methods such as affine transformation and intersection with half-space [SFÁ19]. The union of two state sets is computed at the beginning of the first flowpipe segment or after taking a discrete jump because of a guard in the hybrid automaton. Therefore, the union method does not influence the running time as much as the other methods. However, as it is used at the begining of a flowpipe creation, it means that it is going to be used at least one time. To avoid inefficiency, three options of computation for the union of two star sets will be introduced. All of them have their positive and negative consequences. While trying to find a efficient solution for the computation, we took inspiration from the zonotopes [Gir05], $\mathcal{H}$-polytopes [SFÁ19] and implementation of [TCD$^+$19]. The techniques used for these representations were used together to compute the union of two star sets. As we introduced star sets, similarities between star sets and both of these representations might not be apparent at first glance. The similarity between zonotope and star sets is that they both have a basis and a center and when it comes to the $\mathcal{H}$-polytopes the linear constraints of the star sets can be interpreted as a $\mathcal{H}$-polytope. The options for computing the union of two star sets will be explained in the following separately and after that the advantages and disadvantages of the options will be discussed.

Option 1

Conversion to $\mathcal{H}$-polytope is always a viable option to compute the union of two star sets. As it is already introduced, how to convert a star set to a $\mathcal{H}$-polytope, this option will be straight-forward. One can convert both of the star sets to a $\mathcal{H}$-polytope and then compute the union of these $\mathcal{H}$-polytopes as it is explained in [SFÁ19]. Then the resulting $\mathcal{H}$-polytope can be converted back to a star set. Therefore the resulting star set will be the union of both of the star sets that were given as inputs.

Option 2

Before explaining how this option works, it needs to be said that this option will not always be viable. This option can be used only if one of the star sets is the affine transformed version of the other star set. For simplicity, we will name the first star set as S_1 and the affine transformed one as S_2. Also the first star set S_1 has to have a standard basis while using this option. The reason for this restriction is that, as both star sets are given we do not know the linear transformation matrix that was applied on the basis of S_1. If this restricton holds, we can take the basis of the star set S_2 as the linear transformation matrix and center of the star set S_2 as the offset vector. It should be also noted that if we know the linear transformation matrix and the offset vector applied on the star set S_1 this method can be used without these restrictions.

As seen there are many restrictions about where this option can be used. However, while computing the reachability of a hybrid automaton, at first a star set is created with a standard basis. Because of that this option will be eligible to be used, at least one one time during the reachability analysis. First, the introduction of the union for two $\mathcal{H}$-polytopes will be explained, as we are going to take at first a similar approach:

Let A_H and B_H two n- dimensional $\mathcal{H}$-polytopes. From the definition of convex closure of the union we obtain [SFÁ19]:

$$cl(A \cup B) = \{la + (1-l)b \mid l \in [0,1] \wedge a \in A_H \wedge b \in B_H\}$$

The same logic will be adapted to star sets. Let the star sets be: First star set S_1 is defined as $S_1 = \langle c_1, V_1, P \rangle$ and the second star set S_2 is defined as $S_2 = \langle c_2, V_2, P \rangle$ with $P(\alpha) \triangleq A\alpha \leq d$, $A \in \mathbb{R}^{p \times n}$, $\alpha = [\alpha_1, ..., \alpha_n]^T$, $d \in \mathbb{R}^{p \times 1}$,

V_1,$V_2 \in \mathbb{R}^{n\times n}$, $c_1, c_2 \in \mathbb{R}^n$ and p,$n \in \mathbb{R}$.

Here we assumed that S_2 is a affine transformed version of S_1. Therefore, we also assumed that the predicates of both star sets are equal. *Because affine transformation for star sets does not change the predicates!* Also let the linear transformation matrix be: $L \in \mathbb{R}^{n\times n}$ and the offset vector $b \in R^n$, which are used for affine transformation, which is applied on the star set S_1 to compute S_2. Also $I \in \mathbb{R}^{n\times n}$ is the identity matrix.

The derivation of the convex closure of two star sets:

$$\begin{aligned} cl(S_1 \cup S_2) &= \{lx_1 + (1-l)x_2 \mid l \in [0,1] \wedge x_1 \in S_1 \wedge x_2 \in S_2\} \\ &= \{lx_1 + (1-l)(Lx_1 + b) \mid l \in [0,1] \wedge x_1 \in S_1\} \\ &= \{lx_1 - lLx_1 - lb + Lx_1 + b \mid l \in [0,1] \wedge x_1 \in S_1\} \\ &= \{l(x_1 - b - Lx_1) + Lx_1 + b \mid l \in [0,1] \wedge x_1 \in S_1\} \\ &= \{l((I-L)x_1 - b) + Lx_1 + b \mid l \in [0,1] \wedge x_1 \in S_1\} \end{aligned}$$

This resulting set is the union of the star sets S_1 and S_2. While implementing, it is important to recognize that the part $\{Lx_1 + b \mid x_1 \in S_1\}$ actually represents the star set S_2. From this point on we are going to refer to the set $\{l((I-L)x_1 - b) \mid l \in [0,1] \wedge x_1 \in S_1\}$ as S_3 for simplicity.

It might not be clear, how we are going to handle variable l as $l \in [0,1]$. To show how we will handle the variable l, S_3 will be further derived.

$$\begin{aligned} S_3 &= \{l((I-L)x_1 - b) \mid l \in [0,1] \wedge x_1 \in S_1\} \\ &= \{l((I-L)c_1 + (I-L)\alpha V_1 - b) \mid l \in [0,1]\} \\ &= \{l(I-L)c_1 + (I-L)l\alpha V_1 - lb \mid l \in [0,1]\} \\ &= \{l((I-L)c_1 - b) + (I-L)l\alpha V_1) \mid l \in [0,1]\} \end{aligned}$$

As $(I-L)l\alpha V_1 = (I-L)\alpha V_1$ when $l = 1$. We can discard l in this case. As l can be thought as a scaling factor and when $l = 1$, it is the same set as $(I-L)\alpha V_1$.

$$\begin{aligned} \Rightarrow S_3 &= \{l((I-L)c_1 - b) + (I-L)\alpha V_1 \mid l \in [0,1]\} \\ \Rightarrow cl(S_1 \cup S_2) &= \{l((I-L)c_1 - b) + (I-L)\alpha V_1 + Lx_1 + b \mid l \in [0,1] \wedge x_1 \in S_1\} \\ &= \{c_2 + l((I-L)c_1 - b) + (I-L)\alpha V_1 + \alpha V_2 \mid l \in [0,1]\} \end{aligned}$$

Here we will think $l((I-L)c_1 - b)$ as a star set. Let $S_4 = \langle c_4, V_4, P_4\rangle$ with $c_4 \in \mathbb{R}^n$ being a vector filled with zeros. $V_4 = (I-L)c_1 - b$ and

$P_4(\alpha') \triangleq \begin{pmatrix}-1\\1\end{pmatrix}\alpha' \leq \begin{pmatrix}0\\1\end{pmatrix}$ with $\alpha' \in \mathbb{R}$ and $V_4 \in \mathbb{R}^{n\times 1}$. The predicate is chosen to represent the variable l. After this we will write the convex closure of the union as:

$$\begin{aligned}\Rightarrow cl(S_1 \cup S_2) &= \{c_2 + \alpha V_2 + c_4 + \alpha' V_4 + (I-L)\alpha V_1\}\\ &= \{c_2 + \alpha V_2 + \alpha' V_4 + (I-L)\alpha V_1\}\end{aligned}$$

From the definition of the star set we will create a new star set with the same properties defined here. We choose as the center c_2 from S_2. We propose an approach which we concatenate a the basis vectors V_2, V_4 and $(I-L)V_1$. Also while concatenating we should also build the conjunction of the corresponding linear constraints of them.

In the following algorithm $L = \Theta_2$.generator() is the linear transformation matrix, $b = \Theta_2$.center() is the translation vector and I is the identity matrix with the appropriate dimension. Also it was previously assumed that linear constraints of both star sets were equal.

Algorithm 9: Union with option 2

Input: Star set Θ_1, star set $\Theta_2 = L\Theta_1 + b$

Output: Star set Θ'

// Represents S_4 defined above

1 $\Theta_3 = (I - \Theta_2.\text{generator}())\Theta_1 - \Theta_2.center()$

// Setting the new generator matrix

2 matrix V'$\leftarrow \begin{pmatrix}\Theta_1.\text{generator}() & \Theta_3.\text{center}() & \Theta_2.\text{generator}()\end{pmatrix}$

// 0 represents a matrix filled with zeros with appropriate dimension

// Setting new constraints matrix

3 matrix A'$\leftarrow \begin{pmatrix}\Theta_1.\text{constraints}() & 0\\ 0 & -1\\ 0 & 1\end{pmatrix}$

4 matrix A''$\leftarrow \begin{pmatrix}A' & 0\\ 0 & \Theta_2.\text{constraints}()\end{pmatrix}$

// Setting new limits vector

5 vector d'$\leftarrow \begin{pmatrix}\Theta_1.\text{limits}()\\ 0\\ 1\\ \Theta_2.\text{limits}()\end{pmatrix}$

// Create a new star set with the new variables

6 return starset(Θ_2.center(),V',A'',d')

Option 3

This option is the most general one and can be computed in every situation. In this method it will be explained how to compute the closure of the union of two star sets. This method is inspired by the paper [Gir05] on zonotopes. Also we will consider our constraints for star set as a $\mathcal{H}$-polytope, like it is mentioned before. It is advised to read the option 2 before this option. As both are using the same principles and it is explained more deeply in the option 2.

Let the star sets be:

First star set Θ is defined as:
$\Theta = \langle c, V, P \rangle$,
$P(\alpha) \triangleq A\alpha \leq d$,
$A \in \mathbb{R}^{p \times m}$,
$\alpha = [\alpha_1, ..., \alpha_m]^T$,
$d \in \mathbb{R}^{p \times 1}$, $c \in \mathbb{R}^n$, p,$m \in \mathbb{R}$

Second star set Θ' is defined as:
$\Theta' = \langle c', V', P' \rangle$,
$P'(\alpha') \triangleq A'\alpha' \leq d'$,
$A' \in \mathbb{R}^{p' \times m}$,
$\alpha' = [\alpha'_1, ..., \alpha'_m]^T$,
$d' \in \mathbb{R}^{p' \times 1}$, $c' \in \mathbb{R}^n$, p',$m \in \mathbb{R}$

So that with the definition of closure of the union of the star sets, it follows:

$$\begin{aligned} cl(\Theta \cup \Theta') &= \{lx + (1-l)x' \mid l \in [0,1] \wedge x \in \Theta \wedge x' \in \Theta'\} \\ &= \{lx + x' - lx' \mid l \in [0,1] \wedge x \in \Theta \wedge x' \in \Theta'\} \\ &= \{c' + \alpha' V' + lc + l\alpha V - lc' - l\alpha' V' | P(\alpha) = \top \wedge P'(\alpha') = \top\} \\ &= \{c' + \alpha' V' + l(c - c' + \alpha V - \alpha' V') | P(\alpha) = \top \wedge P'(\alpha') = \top\} \end{aligned}$$

As seen this set is the union of both sets. To compute this set we are going to use a similar approach like the option 2. The algorithm is in the following.

Algorithm 10: Union with option 3

Input: Star set Θ_1, star set Θ_2

Output: Star set Θ'

```
// Setting the new generator matrix
```

1 matrix V'$\leftarrow$
$\begin{pmatrix}\Theta_2\text{.generator()} & \Theta_1\text{.center()-}\Theta_2\text{.center()} & \Theta_1\text{.generator()} & \Theta_2\text{.generator()}\end{pmatrix}$

```
// 0 represents a matrix filled with zeros with appropriate
   dimension
// Setting new constraints matrix
```

2 matrix A'$\leftarrow \begin{pmatrix}\Theta_2\text{.constraints()} & 0\\ 0 & -1\\ 0 & 1\end{pmatrix}$

3 matrix A''$\leftarrow \begin{pmatrix}A' & 0\\ 0 & \Theta_1\text{.constraints()}\end{pmatrix}$

4 matrix A'''$\leftarrow \begin{pmatrix}A'' & 0\\ 0 & -\Theta_2\text{.constraints()}\end{pmatrix}$

```
// Setting new limits vector
```

5 vector d'$\leftarrow \begin{pmatrix}\Theta_2\text{.limits()}\\ 0\\ 1\\ \Theta_1\text{.limits()}\\ \Theta_2\text{.limits()}\end{pmatrix}$

```
// Create a new star set with the new variables
```

6 return starset(Θ_2.center(),V',A''',d')

Discussion

We presented three options of convex hull computation for star sets. In this part we will be discussing their advantages and disadvantages.

The first option, which uses conversion, is the most precise way to compute the convex hull of two star sets. $\mathcal{H}$-polytopes are one of the most precise representations [SFÁ19], however conversion to $\mathcal{H}$-polytope can be costly, while using models with high dimensions. However, this option does not increase the number of basis vectors in the star set. For models with many jumps, this is a great advantage. As after every jump union of the star sets will be computed. The computation of the convex hull with conversion does not increase the number of generators meaning that the intersection with half-space and affine transformations can be computed more efficiently. This approach, empirically speaking, worked well for the

problem *bouncing ball* and also outperformed the other options explained before. However, for higher dimesional problems this method is too inefficient as conversion takes too much time.

The second option is comparable in precision with the first option and more precise than the last option. When using models with high dimensions, this increase in precision can be very important. It also does not increase the constraint matrix size as much as the last option. As the first option is not viable for models with high dimensions, this option can be used in those situations. However, this method can not be used every time, we are computing the convex hull, as it needs the first star set to have a standard basis and the second star set to be the affine transformed version of the first star set. This option is particularly helpful, when using models which are purely dynamical and has high dimensionality (*building*). In such systems the union method is only used once at the beginning of the flow-pipe construction [SFÁ19], when the first star set has a standard basis and the second star set is the affine transformed version of the first star set. This option increases the precision and the efficiency of the computation in such situations.

Third option is the most general one. It can be used on every two star sets, if the dimensions match. This option affects the efficiency of the star sets heavily, as it increases the constraint matrix size, even if it is not used often compared to the other methods such as affine transformation and intersection with half-spaces [SFÁ19]. The union method that is explained, is not a computationally expensive operation but it influences other methods by increasing the constraint matrix size by three times in average. For a hybrid automaton that has multiple guards, it could affect the run time drastically. That being so, the logical solution for problems with low dimensions would be computing the convex hull of two star set by converting them both in $\mathcal{H}$-polytopes and then computing the convex hull of them like it is explained in option 1.

Example

In this part an example of the option 1 and option 2 for convex hull between star sets will be given.
For the star set $\Theta = \langle c, V, P \rangle$*:*

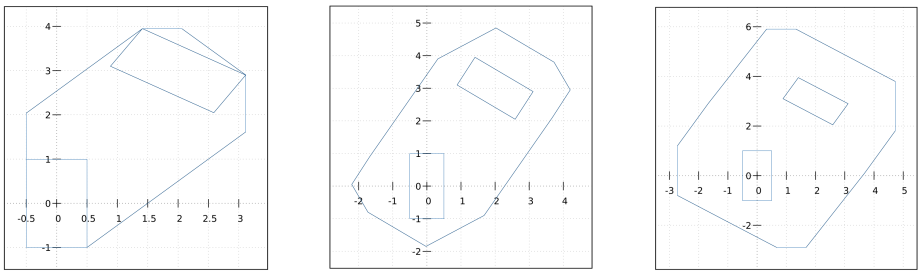
$$\text{Take } V = \{\begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}\} \text{ the unit vectors and } c = (3,3)$$

Consider $P(\bar{\alpha}) = A\bar{\alpha} \leq d$ where $A = \begin{pmatrix} 2 & 0 \\ -2 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}$ and $d = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$

The star set $\Theta = \langle c, V, P \rangle$ *then defines the rectangular set:*

$$[\![\Theta]\!] = \{(x,y) | -0{,}5 \leq x \leq 0{,}5 \wedge -1 \leq y \leq 1\}$$

Let the linear transformation matrix be $L = \begin{pmatrix} cos(45°) & -sin(45°) \\ sin(45°) & cos(45°) \end{pmatrix}$ and the offset vector $b = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$. Let second star set $\Theta_2 = L\Theta + b$. The results, that our implementation in HyPro produces, are given in the following:



(a) Using conversion (b) Using option 2 (c) Using option 3

Figure 3.2: Given for comparison of over-approximation in both options

The reason for the over-approximation in all cases is that, when number of constraints and the basis vectors increase it will be harder to compute the united star set. Also to plot we will use conversion to $\mathcal{V}$-polytopes. Higher the number of generators and constraint matrix size harder it is to be exact, while converting to a $\mathcal{V}$-polytope.

Chapter 4

Experimental Results

Previously, the definition and how to compute the methods needed for reachability analysis of a star set has been introduced. In this chapter we would like to give a performance evaluation of our implementation. This chapter will introduce at first the tests, that we are going to run. Then the experimental setup and the benchmarks in the introduced tests will be explained.

4.1 Setup

HyPro and HyDra. The methods introduced in the chapter before have been implemented in the C++ library HyPro. This library offers reachability analysis for affine hybrid automata. Also as many other state set representations are already implemented in this library, it gives us the opportunity to compare our results with these representations. We also used GLPK as the LP solver library [Mak08]. HyDra is the tool, which is based on the library HyPro. It extends the usability with several concepts, such as partial path refinement, subspace decomposition and paralleziation [18].

Conditions

All tests were done on a Intel core i7-7500U CPU at 2.7GHz with 8 GB RAM plus a 4 GB swap memory. Also for all tests with dimension smaller than 11, we used the option 1, for the union computation as explained before. Timeout(TO) was set to 10 minutes. If a benchmark could not be verified as safe -1 will be entered to the table, else the corresponding time that it completed the benchmark will be entered to the table. While computing the benchmarks, we did not use partial path refinement and subspace de-

composition. Also all only one thread was used during all computations, while GLPK being the LP Solver.

4.2 Benchmarks

Before we experimenting a short introduction for each benchmark will be given.

Bouncing Ball [CSM+15] This example is already introduced in the preliminaries section. To summarize the benchmark, it simulates a ball which is dropped from an initial height.

Building [TNJ16]. This benchmark is purely dynamical, meaning that it has no transitions. This benchmark is not only chosen for its high dimensionality but also to verify it high precision is needed.

Platoon30 [TNJ16]. This benchmark is about three autonomous vehicles all driving in only one lane. It needs to be checked if any of the vehicles collide with each other. This benchmark has been chosen for its medium difficulty, as it needs it needs less precision than the building benchmark and is also less dimensional with twelve variables. However, as it has transitions it offers new challenges.

SRA01(Space Rendezvous) [TNJ16]. This benchmark is chosen for its need for precision. Because it has many invariants and bad state constraints. With this benchmark we want to evaluate if the star sets precision and as this benchmark is in need for precision, checking if the it is satisfied normally takes a long time to complete.

ISS01, ISS02, Heat5[1]. These benchmarks are presented in the competition ARCH 2020. Because of that they are particularly difficult and high dimensional. They all have no transitions and purely dynamical. We would like to make a limit check for star sets, if it can solve these problems, as they are particularly difficult.

The following table presents the most important aspects of the benchmarks we explained.

[1] Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH), cps-vo.org/group/ARCH

Benchmark	**Dim**	**Loc**	**Trans**	**Max Jumps**	δ
BouncingBall	2	1	1	3	0.01
Building	50	1	0	20	0.05
Platoon30	12	2	2	1000	0.1
SRA01	5	3	2	1000	1
ISS01	277	1	0	0	0.001
ISS02	277	1	0	0	0.001
Heat5	123	1	0	0	0.002

Figure 4.1: **Benchmarks**. Notation: Dim=Dimension, Loc=Number of locations, Trans=Number of Transitions, Max Jumps= Number of the jumps the system is allowed to take, δ=Time step-

In the following table the run time evaluation of different representations in comparison to the star sets will be introduced. Before continuing with the run times some notation will be introduced. Timeouts will be marked as "TO". We mark the cell with -1 where safety can not be proven. If the memory of the device we are using is full and can not continue we will mark it with "-2". All running times are given in seconds. It has to be noted that in all benchmark scenarios, we skipped the plotting step. As to plot a star set it needs to be converted to a $\mathcal{H}$-polytope and this operation in high dimensions is inefficent and causes timeouts.

Benchmark	Representation	Run Time
BouncingBall	Star set	0.968
	Support function	0.170
	Box	0.005
Building	Star set	29.182
	Support function	0.317
	Box	-1
Platoon30	Star set	-1
	Support function	-1
	Box	-1
SRA01	Star set	143.367
	Support function	-1
	Box	-1
ISS01	Star set	180.078
	Support function	44.904
	Box	0.832
ISS02	Star set	-1
	Support function	-1
	Box	-1
Heat5	Star set	-2
	Support function	8.908
	Box	-1

Figure 4.2: Running times.

As seen from the table above star sets are able solve most of the tests given. However, as it uses a LP-solver to compute if a half-space is satisfied sometimes it causes memory issues like in Heat5. Because of that the precision of star sets is also dependent on the LP-solver used.

Chapter 5

Conclusion

Summary. First, we gave a short introduction about formal verification of hybrid systems using flowpipe construction. In this work, we mainly examined one of the state-set representations named star set. Furthermore, the operations that are needed for reachability analysis are introduced. For methods that caused inefficiency, such as union, we tried to give other options so that the computation could be faster. Some naive ideas that we need to avoid were shown and the alternatives were explained. As seen from the experimental results section, the star sets are a viable option for most of the benchmarks. It is precise and efficient enough to verify tests such as SRA01. However, its efficiency is mainly dependent on the number of constraints and LP-solvers. Star sets provide a precision near the likes of $\mathcal{H}$-polytopes with having an efficient affine transformation.

Future Work. In all of the work we interpreted the linear constraints as $\mathcal{H}$-polytopes. However, it would be interesting if our predicates would be a box rather than linear constraints. This idea was advised while the thesis was being written and also implemented in HyPro. However, we did not examine this idea thoroughly. It is based upon the affine transformation, which is not applied on the predicates in star sets but on the basis of the star sets. This would mean that the box shape would be rotated without actually changing the predicates, thus no overapproximation will be made for affine transformation on the boxes.

Bibliography

[Ábr15] Erika Ábrahám. Lecture notes on modelling and analyis of hybrid systems. 2015.

[ACH+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1):3–34, 1995.

[BD17a] Stanley Bak and Parasara Sridhar Duggirala. Rigorous simulation-based analysis of linear hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 555–572. Springer, 2017.

[BD17b] Stanley Bak and Parasara Sridhar Duggirala. Simulation-equivalent reachability of large linear systems with inputs. In *International Conference on Computer Aided Verification*, pages 401–420. Springer, 2017.

[CK98] Alongkrit Chutinan and Bruce H Krogh. Computing polyhedral approximations to flow pipes for dynamic systems. In *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No. 98CH36171)*, volume 2, pages 2089–2094. IEEE, 1998.

[CKJ+15] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. Smt-rat: An open source c++ toolbox for strategic and parallel smt solving. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 360–368. Springer, 2015.

[CSM+15] Xin Chen, Stefan Schupp, Ibtissem Ben Makhlouf, Erika Ábrahám, Goran Frehse, and Stefan Kowalewski. A benchmark suite for hybrid systems reachability analysis. In *NASA Formal Methods Symposium*, pages 408–414. Springer, 2015.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[DV16] Parasara Sridhar Duggirala and Mahesh Viswanathan. Parsimonious, simulation based verification of linear systems. In *International Conference on Computer Aided Verification*, pages 477–494. Springer, 2016.

[Gir05] Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *International Workshop on Hybrid Systems: Computation and Control*, pages 291–305. Springer, 2005.

[Mak08] Andrew Makhorin. Glpk (gnu linear programming kit). *http://www. gnu. org/s/glpk/glpk. html*, 2008.

[SFÁ19] Stefan Schupp, Goran Frehse, and Erika Ábrahám. State set representations and their usage in the reachability analysis of hybrid systems. Technical report, Fachgruppe Informatik, 2019.

[TCD+19] Hoang-Dung Tran, Feiyang Cai, Manzanas Lopez Diego, Patrick Musau, Taylor T Johnson, and Xenofon Koutsoukos. Safety verification of cyber-physical systems with reinforcement learning control. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22, 2019.

[TNJ16] Hoang-Dung Tran, Luan Viet Nguyen, and Taylor T Johnson. Large-scale linear systems from order-reduction (benchmark proposal). In *3rd Applied Verification for Continuous and Hybrid Systems Workshop (ARCH), Vienna, Austria*, 2016.

[Tse20] Phillip Tse. Efficient polyhedral state set representations for hybrid systems reachability analysis. 2020.

[Wun96] Roland Wunderling. Paralleler und objektorientierter simplex. 1996.

[18] Erika Ábrahám and Silvia Lizeth Taipa Tarifa, editors. *Proceedings of the PhD Symposium at iFM'18 on Formal Methods: Algorithms, Tools and Applications (PhD-iFM'18)*, volume 483 of *Research report / University of Oslo*. 14th International Conference on integrated Formal Methods, Maynooth (Ireland), 5 Sep 2018 - 7 Sep 2018, Sep 2018. Zweitveröffentlicht auf dem Publikationsserver der RWTH Aachen University 2019.