HyPro: A C++ library of state set representations for hybrid systems reachability analysis

Stefan Schupp

TECHNISCHE UNIVERSITÄT WIEN

TU Wien, Vienna, Austria

May 11, 2022

Hybrid systems

"hybrid: [...] A thing made by combining two different elements."
Oxford dictionary

Hybrid systems are systems combining discrete and continuous behavior.

Hybrid systems

> *"hybrid: [...] A thing made by combining two different elements."*
>
> *Oxford dictionary*

Hybrid systems are systems combining discrete and continuous behavior. They can be found in

- physical processes (bouncing ball, freezing water, ...)
- digital controllers for continuous systems (avionics, automotive, automated plants) $\rightarrow$ cyber-physical systems

As they interact and possibly modify the surrounding environment they are often safety critical.

Hybrid systems reachability analysis

Reachability problem (for hybrid systems)

The reachability problem is the problem to decide whether a state is reachable in a hybrid system from a set of initial states.

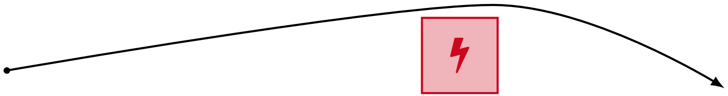
Testing

Hybrid systems reachability analysis

Reachability problem (for hybrid systems)

The reachability problem is the problem to decide whether a state is reachable in a hybrid system from a set of initial states.

Testing

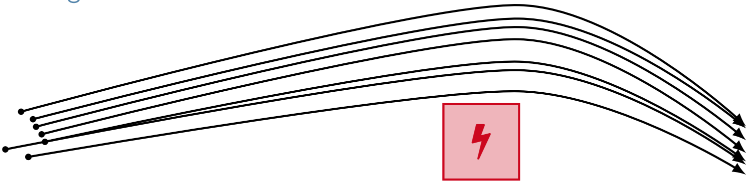


Hybrid systems reachability analysis

Reachability problem (for hybrid systems)

The reachability problem is the problem to decide whether a state is reachable in a hybrid system from a set of initial states.

Testing

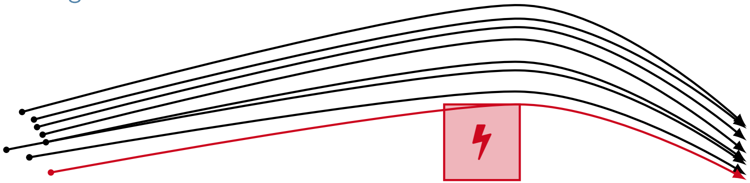


Hybrid systems reachability analysis

Reachability problem (for hybrid systems)

The reachability problem is the problem to decide whether a state is reachable in a hybrid system from a set of initial states.

Testing

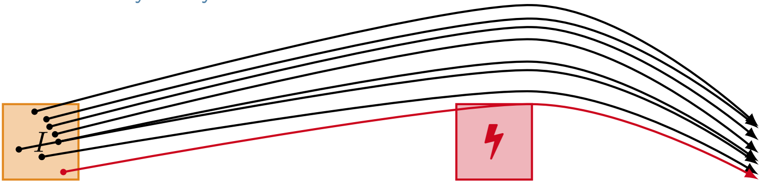


Hybrid systems reachability analysis

Reachability problem (for hybrid systems)

The reachability problem is the problem to decide whether a state is reachable in a hybrid system from a set of initial states.

Reachability analysis

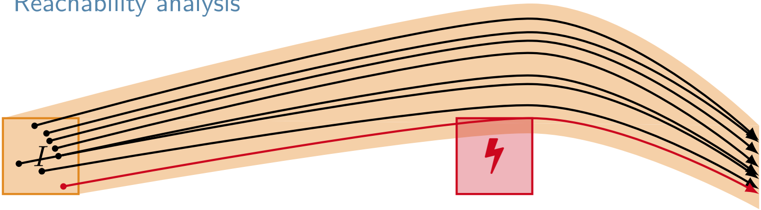


Hybrid systems reachability analysis

Reachability problem (for hybrid systems)

The reachability problem is the problem to decide whether a state is reachable in a hybrid system from a set of initial states.

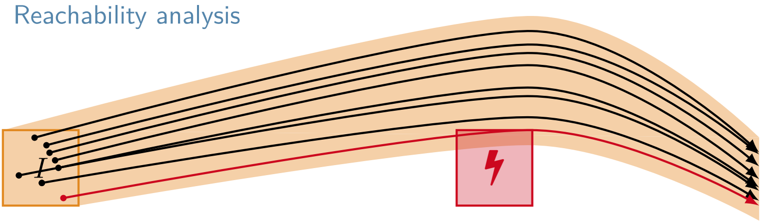
Reachability analysis



Hybrid systems reachability analysis

Reachability problem (for hybrid systems)

The reachability problem is the problem to decide whether a state is reachable in a hybrid system from a set of initial states.



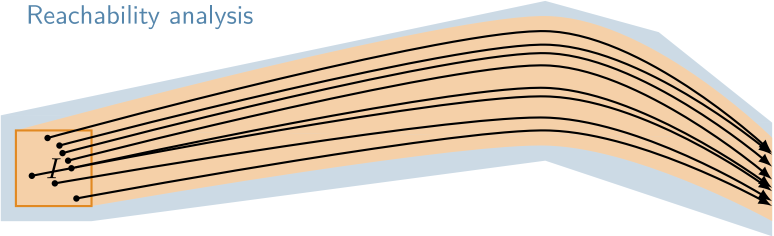
Problem: In general undecidable.

Hybrid systems reachability analysis

Reachability problem (for hybrid systems)

The reachability problem is the problem to decide whether a state is reachable in a hybrid system from a set of initial states.

Reachability analysis

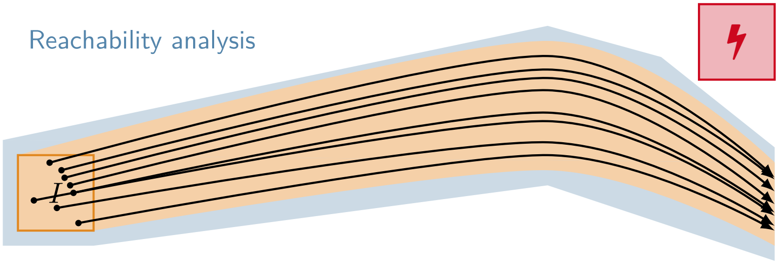


Here: bounded over-approximative reachability analysis for linear hybrid systems.

Hybrid systems reachability analysis

Reachability problem (for hybrid systems)

The reachability problem is the problem to decide whether a state is reachable in a hybrid system from a set of initial states.

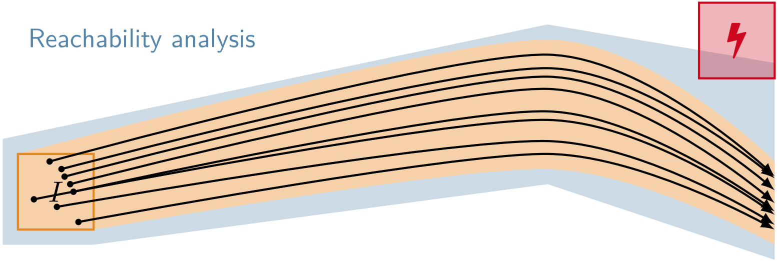


Here: bounded over-approximative reachability analysis for linear hybrid systems.

Hybrid systems reachability analysis

Reachability problem (for hybrid systems)

The reachability problem is the problem to decide whether a state is reachable in a hybrid system from a set of initial states.

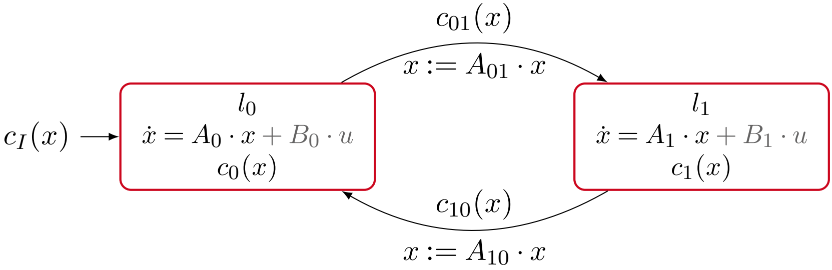


Here: bounded over-approximative reachability analysis for linear hybrid systems.

Hybrid automata

Hybrid systems can be modeled by hybrid automata

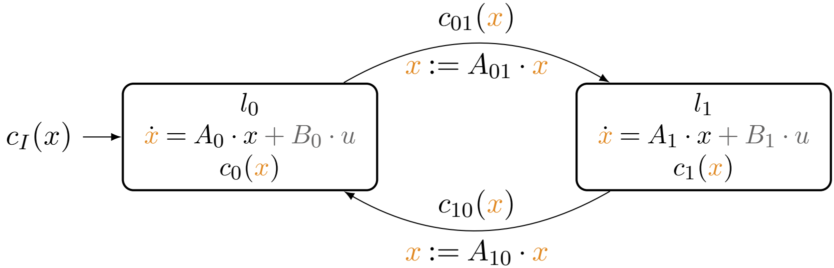
Here: linear hybrid automata



A finite set of locations Loc

Hybrid automata

Hybrid systems can be modeled by hybrid automata
Here: linear hybrid automata

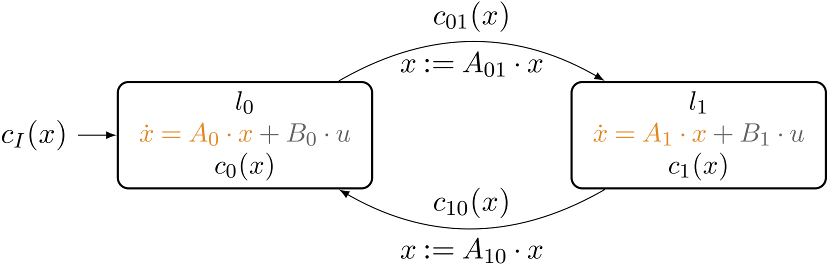


A vector of variables x

Hybrid automata

Hybrid systems can be modeled by hybrid automata

Here: linear hybrid automata

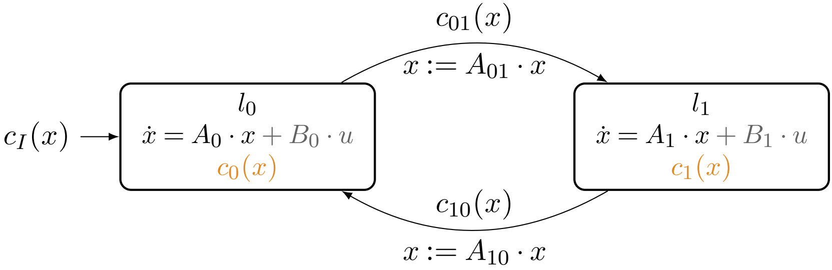


Flow: $Loc \rightarrow Pred_{Var \cup \dot{Var}}$

Hybrid automata

Hybrid systems can be modeled by hybrid automata

Here: linear hybrid automata

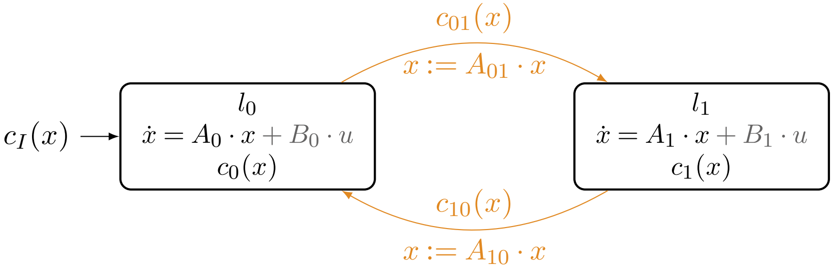


Invariant: $Loc \to Pred_{Var}$

Hybrid automata

Hybrid systems can be modeled by hybrid automata

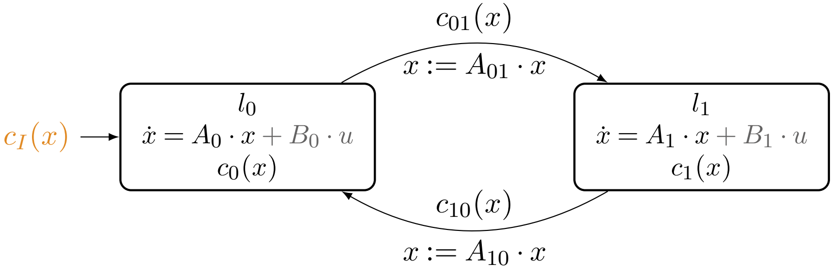
Here: linear hybrid automata



Transitions: $\mathit{Edge} \subseteq \mathit{Loc} \times \mathit{Pred}_{\mathit{Var}} \times \mathit{Pred}_{\mathit{Var} \cup \mathit{Var}'} \times \mathit{Loc}$

Hybrid automata

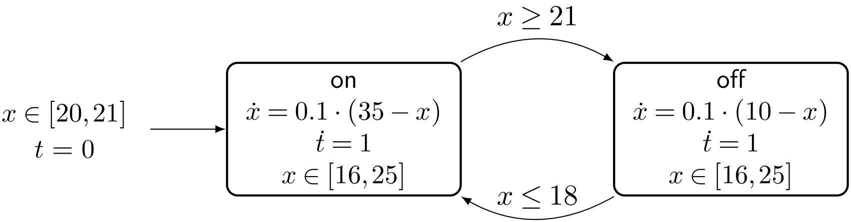
Hybrid systems can be modeled by hybrid automata
Here: linear hybrid automata



An initial set $Loc \to Pred_{Var}$

Hybrid automata – example

Simplified model of a thermostat[1]:



[1] https://www.digitalcity.wien/even-thermostats-have-a-heart/

Reachability analysis algorithm

Basic iterative reachability analysis approach

Input: Set Init of initial states.
Output: Set R of reachable states.

Algorithm:

$$
\begin{array}{l}
R^{\text{new}} := \text{Init}; \\
R := \emptyset; \\
\text{while } (R^{\text{new}} \neq \emptyset)\{ \\
\qquad \begin{array}{ll} R & := R \cup R^{\text{new}}; \\ R^{\text{new}} & := \boxed{\text{Reach}}(R^{\text{new}}) \backslash R; \end{array} \\
\}
\end{array}
$$

Reachability analysis algorithm

Basic iterative reachability analysis approach

Input: Set Init of initial states.
Output: Set R of reachable states.

Algorithm:

$$
\begin{aligned}
&R^{\text{new}} := \text{Init};\\
&R := \emptyset;\\
&\text{while } (R^{\text{new}} \neq \emptyset)\{\\
&\qquad R \qquad := R \cup R^{\text{new}};\\
&\qquad R^{\text{new}} \quad := \boxed{\text{Reach}}(R^{\text{new}}) \setminus R;\\
&\}
\end{aligned}
$$

Question: How to compute Reach for (linear) hybrid systems?

Reachability analysis algorithm

Basic iterative reachability analysis approach

Input: Set Init of initial states.
Output: Set R of reachable states.

Algorithm:

$$
\begin{array}{l}
R^{\text{new}} := \text{Init};\\
R := \emptyset;\\
\text{while } (R^{\text{new}} \neq \emptyset)\{\\
\qquad \begin{array}{ll} R & := R \cup R^{\text{new}};\\ R^{\text{new}} & := \boxed{\text{Reach}}(R^{\text{new}})\backslash R; \end{array}\\
\}
\end{array}
$$

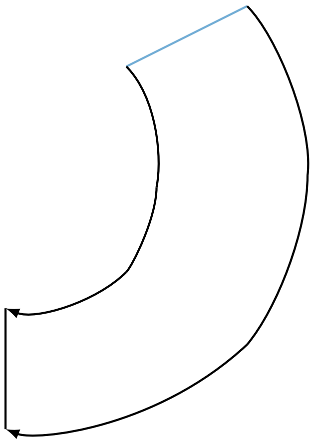
Question: How to compute Reach for (linear) hybrid systems?
Answer: Alternatingly compute time- and jump-successor states.

Linear hybrid automata: Time evolution

- Assume initial set V_0 and flow $\dot{x} = Ax$

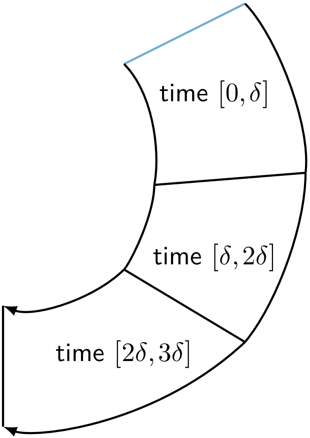
Linear hybrid automata: Time evolution

- Assume initial set V_0 and flow $\dot{x} = Ax$



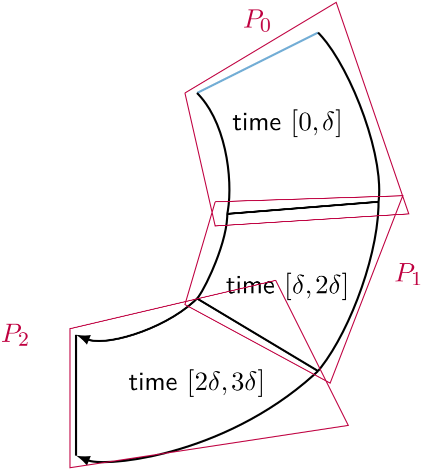
Linear hybrid automata: Time evolution

- Assume initial set V_0 and flow $\dot{x} = Ax$



Linear hybrid automata: Time evolution

- Assume initial set V_0 and flow $\dot{x} = Ax$
- Over-approximate flowpipe segment for time $[i\delta, (i+1)\delta]$ by P_i

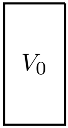


Linear hybrid automata: Time evolution

- Assume initial set V_0 and flow $\dot{x} = Ax$
- Over-approximate flowpipe segment for time $[i\delta, (i+1)\delta]$ by P_i

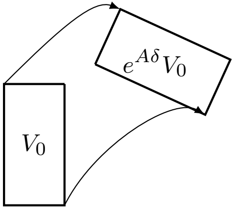
Linear hybrid automata: Time evolution

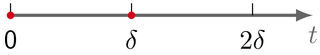
- Assume initial set V_0 and flow $\dot{x} = Ax$
- Over-approximate flowpipe segment for time $[i\delta, (i+1)\delta]$ by P_i



Linear hybrid automata: Time evolution

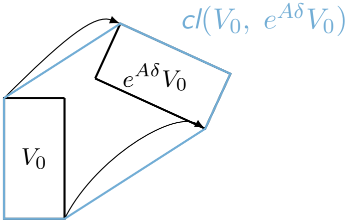
- Assume initial set V_0 and flow $\dot{x} = Ax$
- Over-approximate flowpipe segment for time $[i\delta, (i+1)\delta]$ by P_i





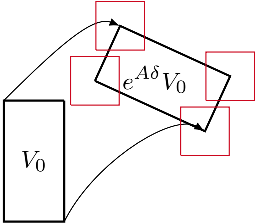
Linear hybrid automata: Time evolution

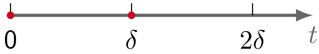
- Assume initial set V_0 and flow $\dot{x} = Ax$
- Over-approximate flowpipe segment for time $[i\delta, (i+1)\delta]$ by P_i



Linear hybrid automata: Time evolution

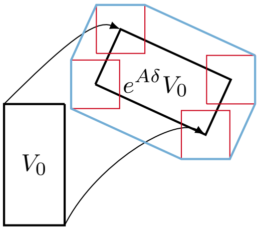
- Assume initial set V_0 and flow $\dot{x} = Ax$
- Over-approximate flowpipe segment for time $[i\delta, (i+1)\delta]$ by P_i





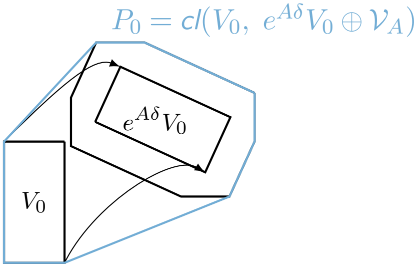
Linear hybrid automata: Time evolution

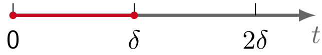
- Assume initial set V_0 and flow $\dot{x} = Ax$
- Over-approximate flowpipe segment for time $[i\delta, (i+1)\delta]$ by P_i



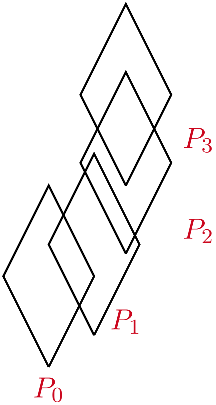
Linear hybrid automata: Time evolution

- Assume initial set V_0 and flow $\dot{x} = Ax$
- Over-approximate flowpipe segment for time $[i\delta, (i+1)\delta]$ by P_i

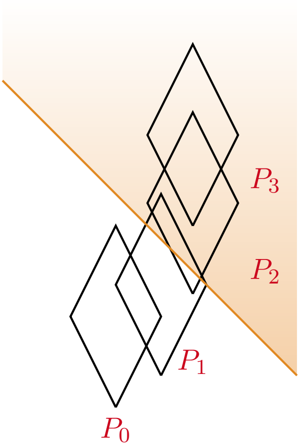




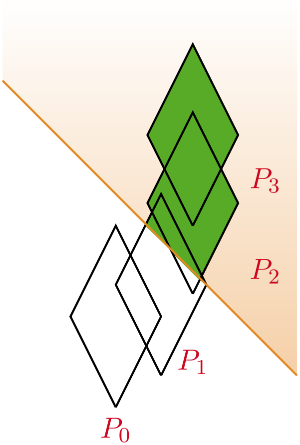
Linear hybrid automata: Discrete steps (jumps)



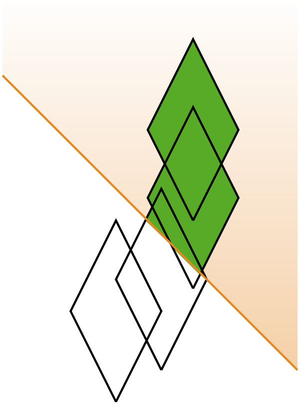
Linear hybrid automata: Discrete steps (jumps)



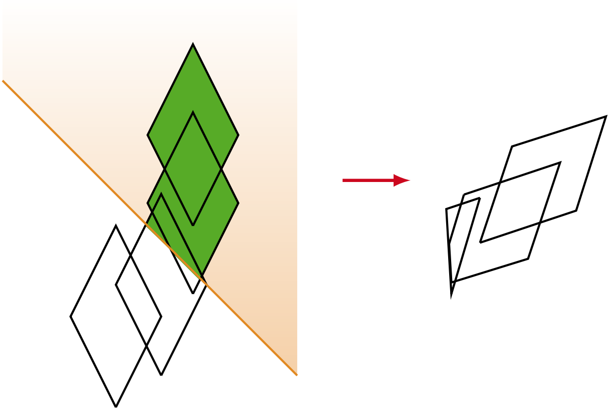
Linear hybrid automata: Discrete steps (jumps)



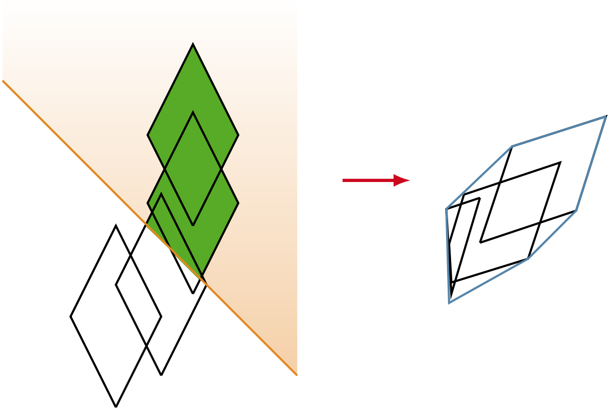
Linear hybrid automata: Discrete steps (jumps)



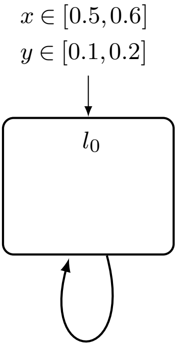
Linear hybrid automata: Discrete steps (jumps)

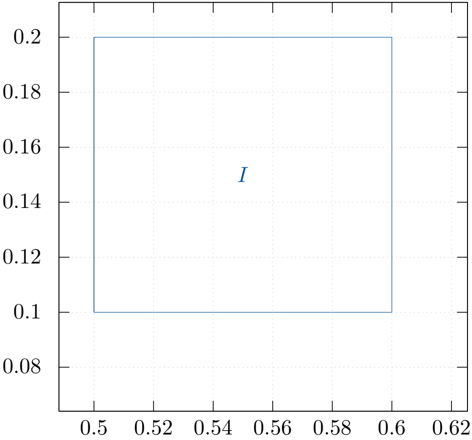


Linear hybrid automata: Discrete steps (jumps)

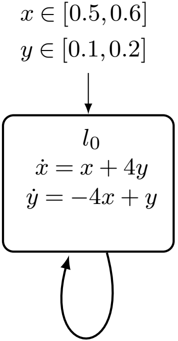


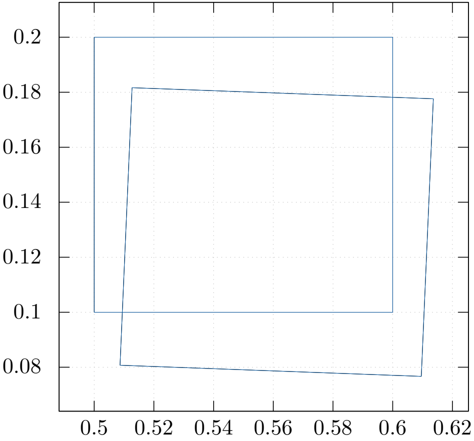
Example - linear hybrid automata





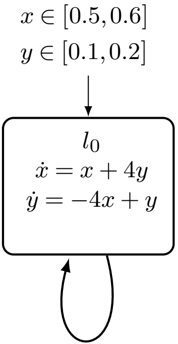
Example - linear hybrid automata

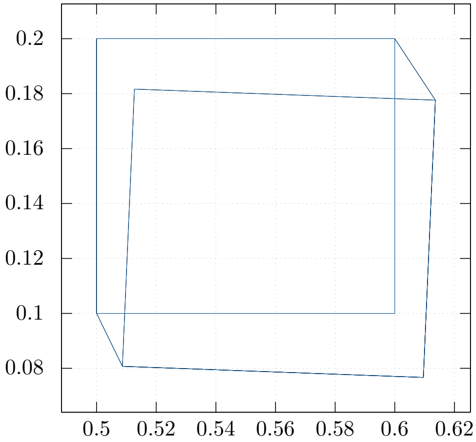




linear transformation: $e^{\delta A} \cdot I$

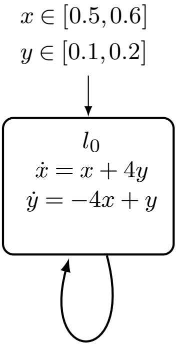
Example - linear hybrid automata

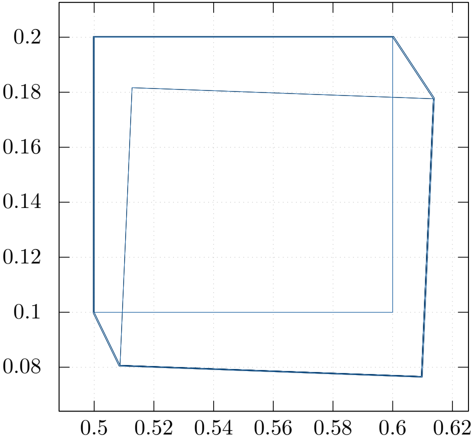




`union`

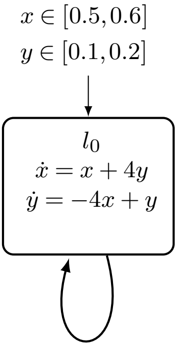
Example - linear hybrid automata

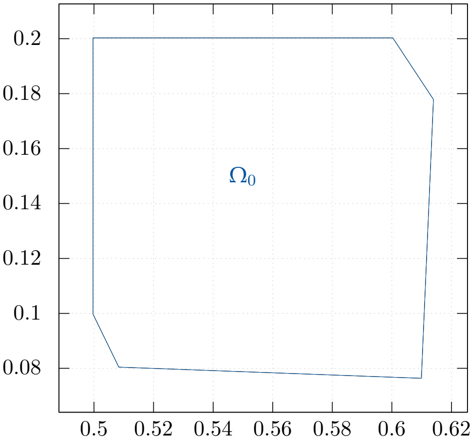




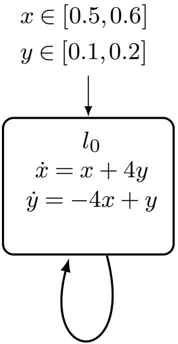
Minkowski sum

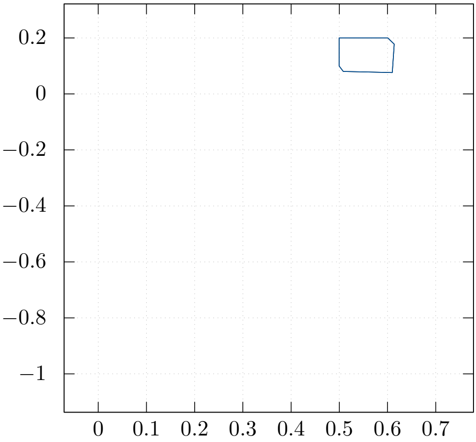
Example - linear hybrid automata



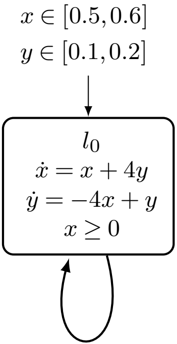


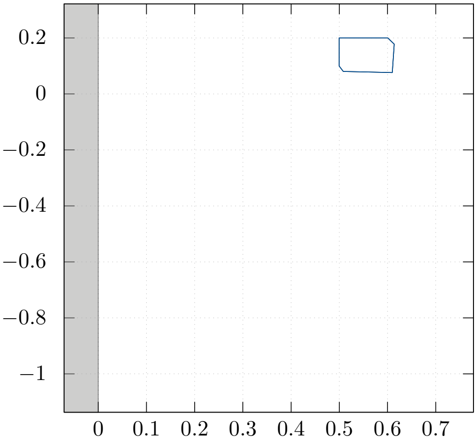
Example - linear hybrid automata





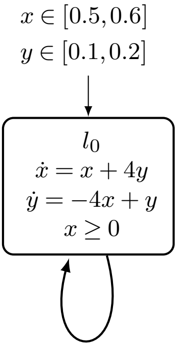
Example - linear hybrid automata

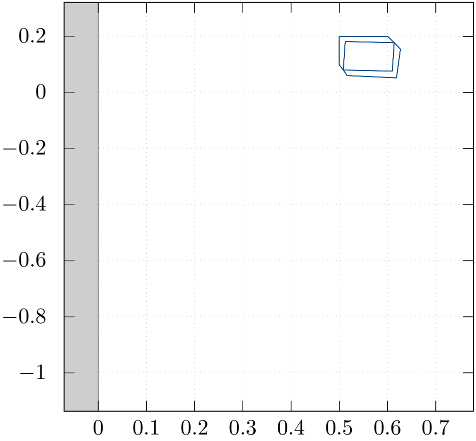




`intersection:` $Inv(l_0) \cap \Omega_i$

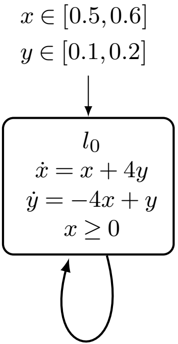
Example - linear hybrid automata

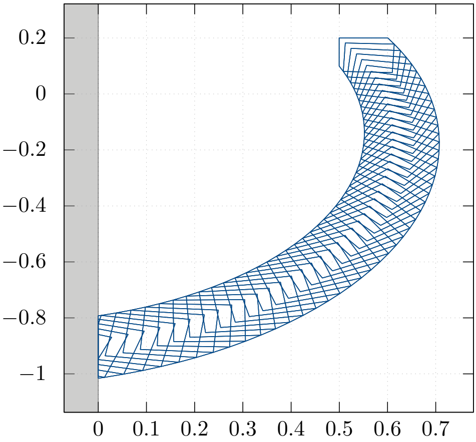




`linear transformation:` $\Omega_{i+1} = e^{\delta A} \cdot \Omega_i$

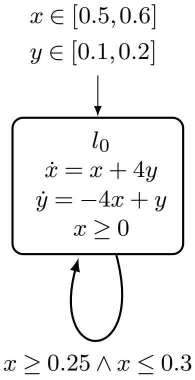
Example - linear hybrid automata

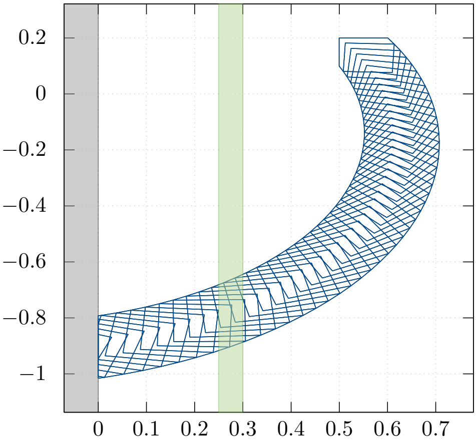




`linear transformation:` $\Omega_{i+1} = e^{\delta A} \cdot \Omega_i$

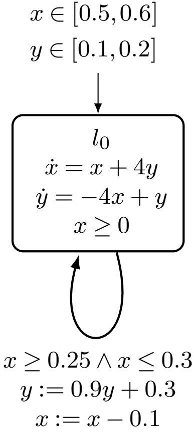
Example - linear hybrid automata

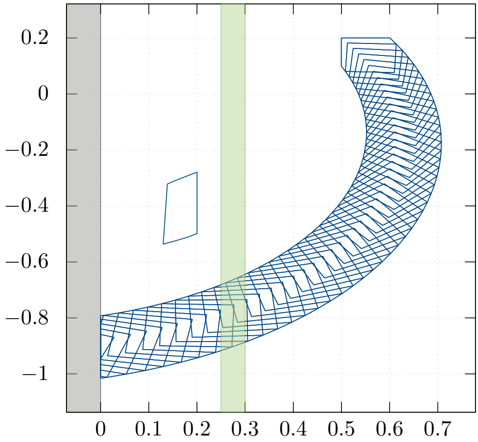




`intersection:` $guard \cap \Omega_i$

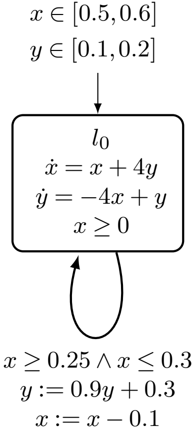
Example - linear hybrid automata

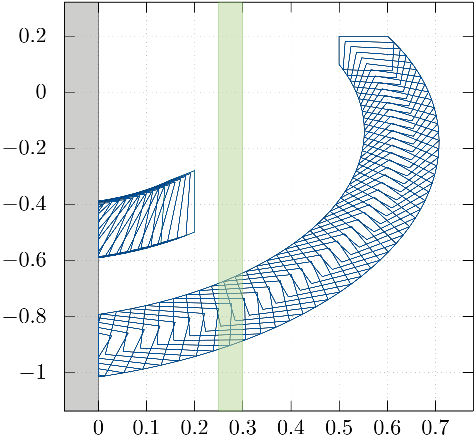




`linear transformation:` $I' := reset(\Omega_i)$

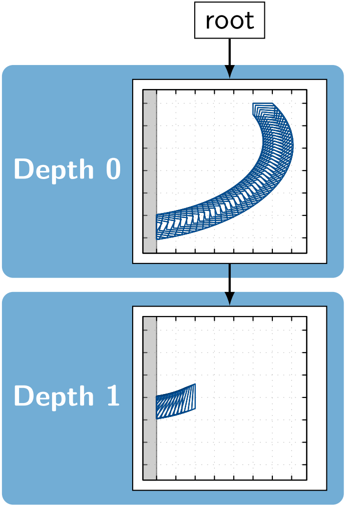
Example - linear hybrid automata





`linear transformation:` $\Omega_{i+1} = e^{\delta A} \cdot \Omega_i$

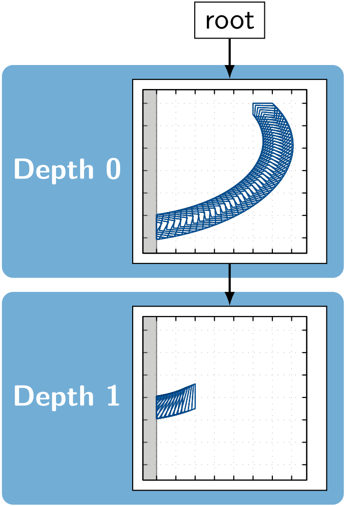
Induced search tree



Induced search tree

The induced search tree depends on:

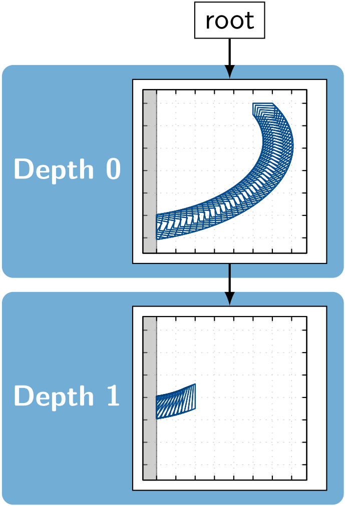
- The model itself
- Bounds (jump depth, time horizon)



Induced search tree

The induced search tree depends on:

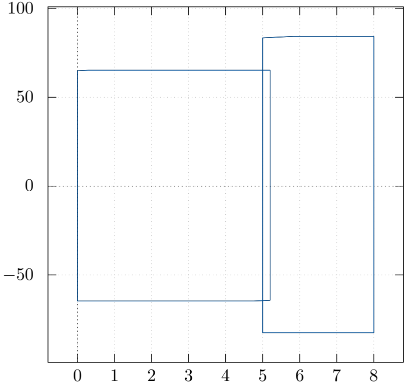
- The model itself
- Bounds (jump depth, time horizon)
- Time step size
- State set representation
- Aggregation settings



Analysis parameters – examples

The *precision* and *running time* depends on several parameters, e.g.,

- Time step size δ

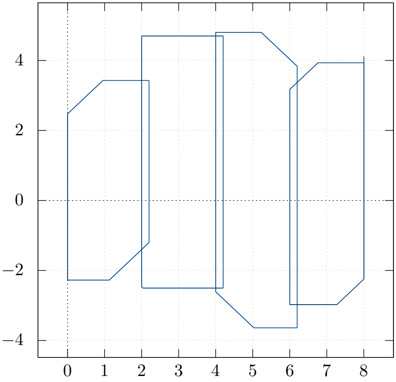


$$\delta = 5$$

Analysis parameters – examples

The *precision* and *running time* depends on several parameters, e.g.,

- Time step size δ

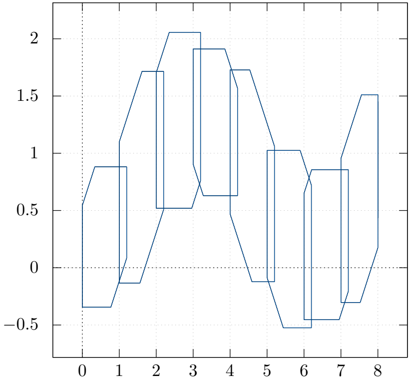


$\delta = 2$

Analysis parameters – examples

The *precision* and *running time* depends on several parameters, e.g.,

- Time step size δ

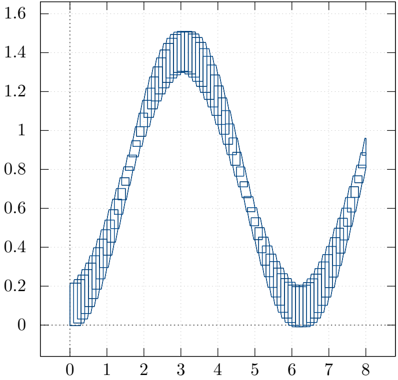


$$\delta = 1$$

Analysis parameters – examples

The *precision* and *running time* depends on several parameters, e.g.,

- Time step size δ

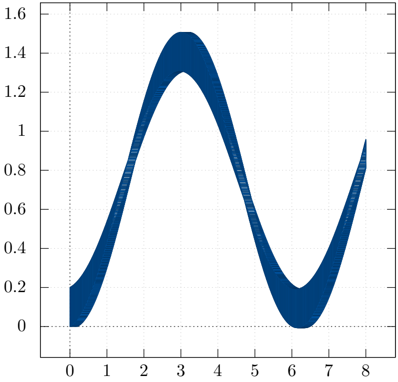


$$\delta = 0.1$$

Analysis parameters – examples

The *precision* and *running time* depends on several parameters, e.g.,

- Time step size δ



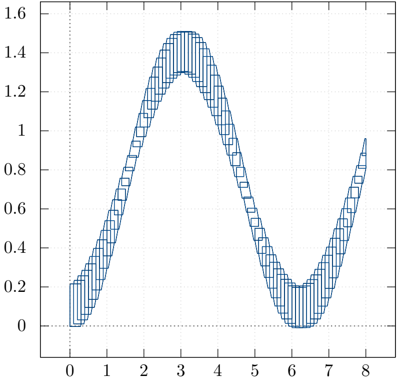
$\delta = 0.01$

TECHNISCHE UNIVERSITÄT WIEN

Analysis parameters – examples

The *precision* and *running time* depends on several parameters, e.g.,

- Time step size δ
- State set representation

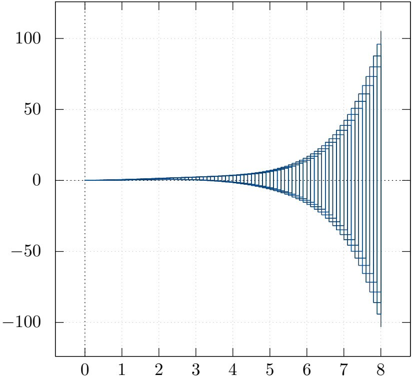


$\delta = 0.1$, *support functions*

Analysis parameters – examples

The *precision* and *running time* depends on several parameters, e.g.,

- Time step size δ
- State set representation

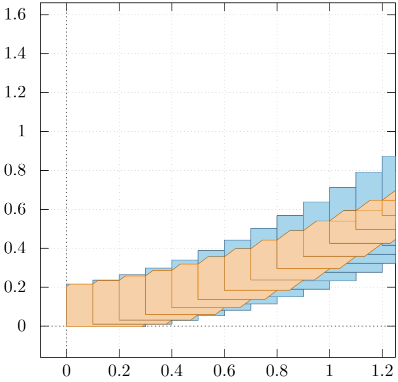


$\delta = 0.1$, *boxes*

Analysis parameters – examples

The *precision* and *running time* depends on several parameters, e.g.,

- Time step size δ
- State set representation

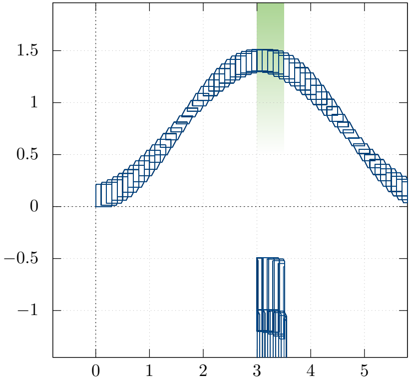


$\delta = 0.1$, *boxes*

Analysis parameters – examples

The *precision* and *running time* depends on several parameters, e.g.,

- Time step size δ
- State set representation
- Clustering/aggregation
    - Default behavior
        - \+ No additional effort
        - − No control of number of discrete successors

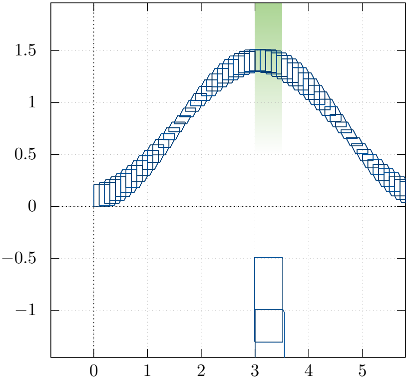


$\delta = 0.1$, support functions, *no aggregation*

Analysis parameters – examples

The *precision* and *running time* depends on several parameters, e.g.,

- Time step size δ
- State set representation
- Clustering/aggregation
  - Default behavior
    - \+ No additional effort
    - − No control of number of discrete successors
  - Aggregation
    - \+ Only one discrete successor
    - − Additional over-approximation



$\delta = 0.1$, support functions, *aggregation*

Sets & required set operations

Required: State set representation.

Problem: There are several ways to represent sets (see next slides).

Sets & required set operations

Required: State set representation.
Problem: There are several ways to represent sets (see next slides).

Required operations on sets:

- `linear transformation` (time successors, reset functions)
- `intersection` (invariants, guards, bad states)
- `union` (first segment, clustering/aggregation)
- `Minkowski sum` (first segment, bloating)

Sets & required set operations

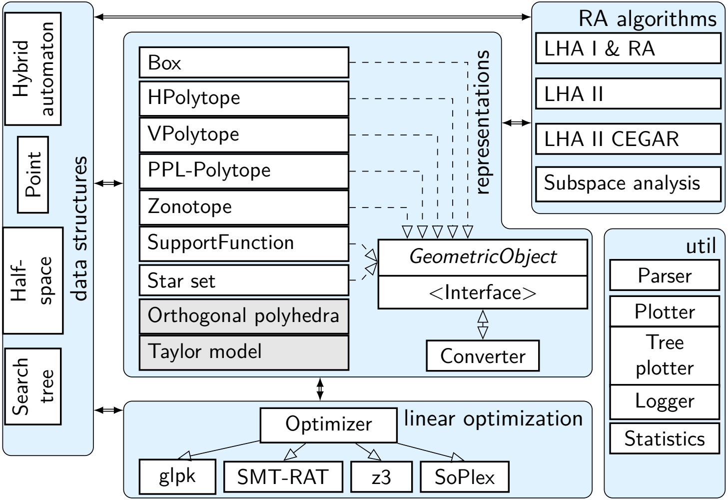
Required: State set representation.
Problem: There are several ways to represent sets (see next slides).

Required operations on sets:

- `linear transformation` (time successors, reset functions)
- `intersection` (invariants, guards, bad states)
- `union` (first segment, clustering/aggregation)
- `Minkowski sum` (first segment, bloating)

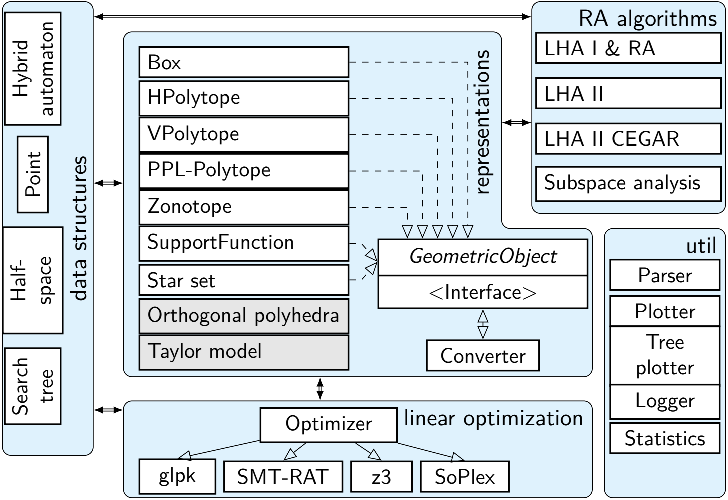
Goal: Unify available state set representations with a common interface.

HyPro[2]



[2] [SÁBMK17]

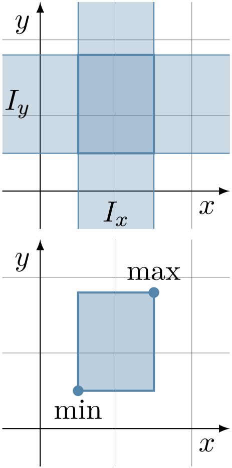
HyPro[2]



[2] [SÁBMK17]

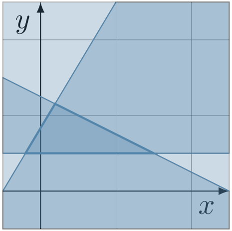
Implemented state set representations

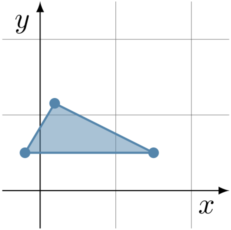
- boxes [MKC09]



Implemented state set representations

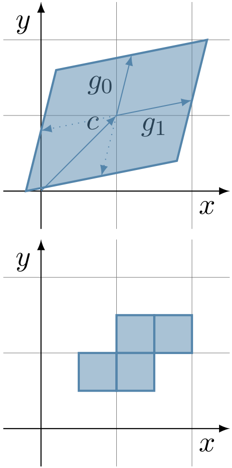
- boxes [MKC09]
- convex polytopes [Zie95]





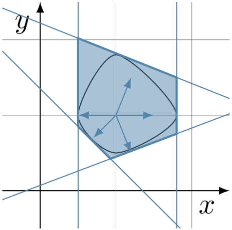
Implemented state set representations

- boxes [MKC09]
- convex polytopes [Zie95]
- zonotopes [Gir05]
- orthogonal polyhedra [BMP99]



Implemented state set representations

- boxes [MKC09]
- convex polytopes [Zie95]
- zonotopes [Gir05]
- orthogonal polyhedra [BMP99]
- support functions [LGG10]
- Taylor models [CÁS12]



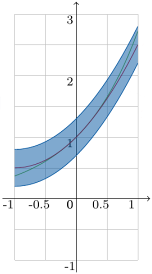


Image: Xin Chen

GeometricObjectBase interface

Set operations:

`X.affineTransformation(matrix A, vector b)`	$AX + b$
`X.minkowskiSum(geometricObject Y)`	$X \oplus Y$
`X.intersectHalfspaces(matrix A, vector b)`	$X \cap \{y \mid Ay \leq b\}$
`X.satisfiesHalfspaces(matrix A, vector b)`	$X \cap \{y \mid Ay \leq b\} \neq \emptyset$
`X.unite(geometricObject Y)`	$cl(X \cup Y)$

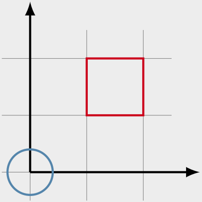
GeometricObjectBase interface

Set operations:

`X.affineTransformation(matrix A, vector b)`	$AX + b$
`X.minkowskiSum(geometricObject Y)`	$X \oplus Y$
`X.intersectHalfspaces(matrix A, vector b)`	$X \cap \{y \mid Ay \leq b\}$
`X.satisfiesHalfspaces(matrix A, vector b)`	$X \cap \{y \mid Ay \leq b\} \neq \emptyset$
`X.unite(geometricObject Y)`	$cl(X \cup Y)$

Recap: Minkowski sum (dilation)

$A \oplus B = \{x \mid x = a + b, a \in A, b \in B\}$



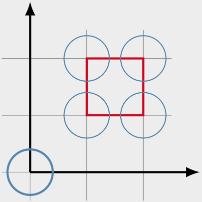
GeometricObjectBase interface

Set operations:

`X.affineTransformation(matrix A, vector b)`	$AX + b$
`X.minkowskiSum(geometricObject Y)`	$X \oplus Y$
`X.intersectHalfspaces(matrix A, vector b)`	$X \cap \{y \mid Ay \leq b\}$
`X.satisfiesHalfspaces(matrix A, vector b)`	$X \cap \{y \mid Ay \leq b\} \neq \emptyset$
`X.unite(geometricObject Y)`	$cl(X \cup Y)$

Recap: Minkowski sum (dilation)

$A \oplus B = \{x \mid x = a + b, a \in A, b \in B\}$



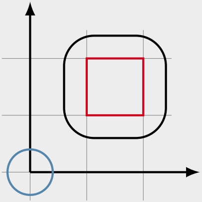
GeometricObjectBase interface

Set operations:

`X.affineTransformation(matrix A, vector b)`	$AX + b$
`X.minkowskiSum(geometricObject Y)`	$X \oplus Y$
`X.intersectHalfspaces(matrix A, vector b)`	$X \cap \{y \mid Ay \leq b\}$
`X.satisfiesHalfspaces(matrix A, vector b)`	$X \cap \{y \mid Ay \leq b\} \neq \emptyset$
`X.unite(geometricObject Y)`	$cl(X \cup Y)$

Recap: Minkowski sum (dilation)

$A \oplus B = \{x \mid x = a + b, a \in A, b \in B\}$



GeometricObjectBase interface

Set operations:

X.affineTransformation(matrix A, vector b)	$AX + b$
X.minkowskiSum(geometricObject Y)	$X \oplus Y$
X.intersectHalfspaces(matrix A, vector b)	$X \cap \{y \mid Ay \leq b\}$
X.satisfiesHalfspaces(matrix A, vector b)	$X \cap \{y \mid Ay \leq b\} \neq \emptyset$
X.unite(geometricObject Y)	$cl(X \cup Y)$

Set utility functions:

- dimension()
- empty()
- vertices()
- project(vector<dimensions> d)
- contains(point p)
- conversion operations
- reduction functions

Operations – complexity

Computational effort required for the most commonly used operations for different representations:

	$\cdot \bigcup \cdot$	$\cdot \bigcap \cdot$	$\cdot \oplus \cdot$	$A(\cdot)$
Box			$+$	
$\mathcal{H}$-polytope	-	$+$	-	-
$\mathcal{V}$-polytope	$+$	-	$+$	$+$
Zonotope			$+$	$+$
Support function	$+$	-	$+$	$+$

Operations – complexity

Computational effort required for the most commonly used operations for different representations:

	$\cdot \bigcup \cdot$	$\cdot \bigcap \cdot$	$\cdot \oplus \cdot$	$A(\cdot)$
Box			+	
$\mathcal{H}$-polytope	-	+	-	-
$\mathcal{V}$-polytope	+	-	+	+
Zonotope			+	+
Support function	+	-	+	+

$\rightarrow$ There is no "perfect" state set representation.

Operations – complexity

Computational effort required for the most commonly used operations for different representations:

	$\cdot \bigcup \cdot$	$\cdot \bigcap \cdot$	$\cdot \oplus \cdot$	$A(\cdot)$
Box			$+$	
$\mathcal{H}$-polytope	-	$+$	-	-
$\mathcal{V}$-polytope	$+$	-	$+$	$+$
Zonotope			$+$	$+$
Support function	$+$	-	$+$	$+$

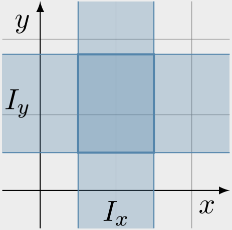
$\rightarrow$ There is no "perfect" state set representation.

Boxes

Boxes are one of the simplest ways to represent a set:

Definition: box [MKC09]

A box $\mathcal{B}$ of dimension n is defined as an ordered vector of intervals



$$\mathcal{B} = (I_0, \ldots, I_n), I_i \in \mathbb{I}$$

Where $\mathbb{I}$ is the set of all real-valued intervals

$$I_i = \{x \mid l \leq x \leq u\} \; l, u \in \mathbb{R},$$

we write $I_i = [l, u] \in \mathbb{I}$

Boxes – operations

Intersection:

$$\mathcal{B}_c = \mathcal{B}_a \cap \mathcal{B}_b = \{x \mid x \in \mathcal{B}_a \wedge x \in \mathcal{B}_b\}$$

Boxes – operations

Intersection:

$$\mathcal{B}_c = \mathcal{B}_a \cap \mathcal{B}_b = \{x \mid x \in \mathcal{B}_a \wedge x \in \mathcal{B}_b\}$$

For boxes:

$$\mathcal{B}_c = I_{a_0} \cap I_{b_0}, \ldots, I_{a_n} \cap I_{b_n}$$

Boxes – operations

Intersection with a half-space (e.g. guards, invariants):

Recap: half-space

A half-space $\mathcal{H} \in \mathbb{R}^n$ contains all points

$$\mathcal{H} = \{x \mid \vec{c}^T \cdot x \leq d,\ \vec{c} \in \mathbb{R}^n,\ d \in \mathbb{R}\}$$

Example:

$$\mathcal{H} = \left\{ x \,\middle|\, \begin{pmatrix} 1 \\ 1 \end{pmatrix}^T \cdot x \leq 1.5 \right\}$$

Excursion: Interval Arithmetic[1]

Binary operations (general case):

$$X \odot Y = \{x \odot y \mid x \in X, y \in Y\}, X, Y \in \mathbb{I}$$

Example (Basic arithmetic operations)

Addition: $[4,5] \quad + \quad [-1,2]$

[1] See e.g., [MKC09] for details.

Excursion: Interval Arithmetic[1]

Binary operations (general case):

$$X \odot Y = \{x \odot y \mid x \in X, y \in Y\}, X, Y \in \mathbb{I}$$

Example (Basic arithmetic operations)

$$\begin{array}{lcccl} \text{Addition:} & [4,5] & + & [-1,2] & = [3,7] \\ \text{Subtraction :} & [4,5] & - & [-1,2] & \end{array}$$

[1] See e.g., [MKC09] for details.

Excursion: Interval Arithmetic[1]

Binary operations (general case):

$$X \odot Y = \{x \odot y \mid x \in X, y \in Y\}, X, Y \in \mathbb{I}$$

Example (Basic arithmetic operations)

$$\begin{array}{lcccl} \text{Addition:} & [4,5] & + & [-1,2] & = [3,7] \\ \text{Subtraction :} & [4,5] & - & [-1,2] & = [2,6] \\ \text{Multiplication:} & [4,5] & \cdot & [-1,2] & \end{array}$$

[1] See e.g., [MKC09] for details.

Excursion: Interval Arithmetic[1]

Binary operations (general case):

$$X \odot Y = \{x \odot y \mid x \in X, y \in Y\}, X, Y \in \mathbb{I}$$

Example (Basic arithmetic operations)

Addition:	$[4,5]$	$+$	$[-1,2]$	$=[3,7]$
Subtraction :	$[4,5]$	$-$	$[-1,2]$	$=[2,6]$
Multiplication:	$[4,5]$	$\cdot$	$[-1,2]$	$=[-5,10]$
Division:	$[4,5]$	$\div$	$[2,3]$	

[1] See e.g., [MKC09] for details.

Excursion: Interval Arithmetic[1]

Binary operations (general case):

$$X \odot Y = \{x \odot y \mid x \in X, y \in Y\}, X, Y \in \mathbb{I}$$

Example (Basic arithmetic operations)

Addition:	$[4,5]$	$+$	$[-1,2]$	$= [3,7]$
Subtraction :	$[4,5]$	$-$	$[-1,2]$	$= [2,6]$
Multiplication:	$[4,5]$	$\cdot$	$[-1,2]$	$= [-5,10]$
Division:	$[4,5]$	$\div$	$[2,3]$	$= [\frac{4}{3}, \frac{5}{2}]$

[1] See e.g., [MKC09] for details.

Excursion: Interval Arithmetic[1]

Binary operations (general case):

$$X \odot Y = \{x \odot y \mid x \in X, y \in Y\}, X, Y \in \mathbb{I}$$

Example (Basic arithmetic operations)

Addition:	$[4,5]$	$+$	$[-1,2]$	$= [3,7]$
Subtraction :	$[4,5]$	$-$	$[-1,2]$	$= [2,6]$
Multiplication:	$[4,5]$	$\cdot$	$[-1,2]$	$= [-5,10]$
Division:	$[4,5]$	$\div$	$[2,3]$	$= [\frac{4}{3}, \frac{5}{2}]$

Corner case: $X \div Y$ with $X, Y \in \mathbb{I}, 0 \in Y$ $\rightarrow$ may cause a split.

[1] See e.g., [MKC09] for details.

Excursion: Interval Arithmetic[1]

Binary operations (general case):

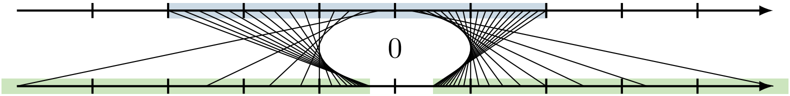
$$X \odot Y = \{x \odot y \mid x \in X, y \in Y\}, X, Y \in \mathbb{I}$$

Example (Basic arithmetic operations)

Addition:	$[4,5]$	$+$	$[-1,2]$	$=[3,7]$
Subtraction :	$[4,5]$	$-$	$[-1,2]$	$=[2,6]$
Multiplication:	$[4,5]$	$\cdot$	$[-1,2]$	$=[-5,10]$
Division:	$[4,5]$	$\div$	$[2,3]$	$=[\frac{4}{3},\frac{5}{2}]$

Corner case: $X \div Y$ with $X, Y \in \mathbb{I}, 0 \in Y$ $\rightarrow$ may cause a split.

Example: $[1,1] \div [-3,2]$



[1]See e.g., [MKC09] for details.

ICP-style Half-space Intersection

Interval constraint propagation (ICP):

- Often used in SMT as a theory solver
- In general incomplete
- Exploits interval arithmetic

ICP-style Half-space Intersection

Interval constraint propagation (ICP):

- Often used in SMT as a theory solver
- In general incomplete
- Exploits interval arithmetic

Example: Encoding of inequalities for interval-valued variables x, y with intervals $I_x, I_y \in \mathbb{I}$:

$$Sat(x + 2 \cdot y \leq 17) = I_x + 2 \cdot I_y \cap (-\infty, 17]$$

ICP-style Half-space Intersection

Interval constraint propagation (ICP):

- Often used in SMT as a theory solver
- In general incomplete
- Exploits interval arithmetic

Example: Encoding of inequalities for interval-valued variables x, y with intervals $I_x, I_y \in \mathbb{I}$:

$$Sat(x + 2 \cdot y \leq 17) = I_x + 2 \cdot I_y \cap (-\infty, 17]$$

Approach: Given c: $\sum a_i \cdot x_i \sim d$ with x_i interval-valued

- For each variable x_i with interval $[a, b]$:
  - Solve c for x_i (symbolically) to get c'

ICP-style Half-space Intersection

Interval constraint propagation (ICP):

- Often used in SMT as a theory solver
- In general incomplete
- Exploits interval arithmetic

Example: Encoding of inequalities for interval-valued variables x, y with intervals $I_x, I_y \in \mathbb{I}$:

$$Sat(x + 2 \cdot y \leq 17) = I_x + 2 \cdot I_y \cap (-\infty, 17]$$

Approach: Given c: $\sum a_i \cdot x_i \sim d$ with x_i interval-valued

- For each variable x_i with interval $[a, b]$:
  - Solve c for x_i (symbolically) to get c'
  - Substitute intervals for all $x_j, j \neq i$ in c', solve to get interval $[a', b']$

ICP-style Half-space Intersection

Interval constraint propagation (ICP):

- Often used in SMT as a theory solver
- In general incomplete
- Exploits interval arithmetic

Example: Encoding of inequalities for interval-valued variables x, y with intervals $I_x, I_y \in \mathbb{I}$:

$$Sat(x + 2 \cdot y \leq 17) = I_x + 2 \cdot I_y \cap (-\infty, 17]$$

Approach: Given c: $\sum a_i \cdot x_i \sim d$ with x_i interval-valued

- For each variable x_i with interval $[a, b]$:
  - Solve c for x_i (symbolically) to get c'
  - Substitute intervals for all $x_j, j \neq i$ in c', solve to get interval $[a', b']$
  - Update interval for $x_i \in [a, b] \cap [a', b']$

ICP-style Half-space Intersection

Interval constraint propagation (ICP):

- Often used in SMT as a theory solver
- In general incomplete
- Exploits interval arithmetic

Example: Encoding of inequalities for interval-valued variables x, y with intervals $I_x, I_y \in \mathbb{I}$:

$$Sat(x + 2 \cdot y \leq 17) = I_x + 2 \cdot I_y \cap (-\infty, 17]$$

Approach: Given c: $\sum a_i \cdot x_i \sim d$ with x_i interval-valued

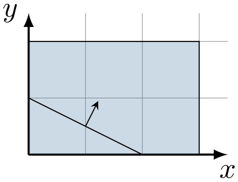
- For each variable x_i with interval $[a, b]$:
  - Solve c for x_i (symbolically) to get c'
  - Substitute intervals for all $x_j, j \neq i$ in c', solve to get interval $[a', b']$
  - Update interval for $x_i \in [a, b] \cap [a', b']$

If one interval becomes empty, the constraint is not satisfiable.

ICP-style Half-space Intersection: Example

Example

Assume $\mathcal{B} = [0,3] \times [0,2]$ and a constraint $c\colon x + 2 \cdot y \leq 2$.



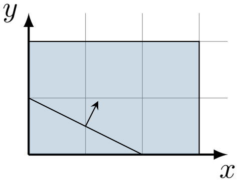
[2] See [Sch19] for a proof.

ICP-style Half-space Intersection: Example

Example

Assume $\mathcal{B} = [0,3] \times [0,2]$ and a constraint c: $x + 2 \cdot y \leq 2$.

Contraction for x:



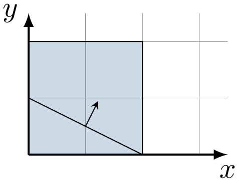
[2] See [Sch19] for a proof.

ICP-style Half-space Intersection: Example

Example

Assume $\mathcal{B} = [0,3] \times [0,2]$ and a constraint c: $x + 2 \cdot y \leq 2$.

Contraction for x: $x \leq 2 - 2 \cdot y \Leftrightarrow x \in [0,3] \cap (-\infty, 2] - [0,4] \rightarrow x \in [0,2]$



[2]See [Sch19] for a proof.

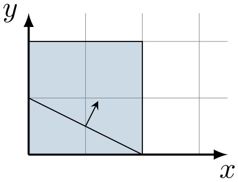
ICP-style Half-space Intersection: Example

Example

Assume $\mathcal{B} = [0,3] \times [0,2]$ and a constraint c: $x + 2 \cdot y \leq 2$.

Contraction for x: $x \leq 2 - 2 \cdot y \Leftrightarrow x \in [0,3] \cap (-\infty, 2] - [0,4] \rightarrow x \in [0,2]$

Contraction for y:



[2] See [Sch19] for a proof.

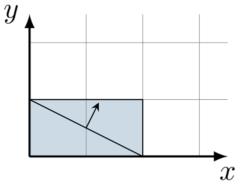
ICP-style Half-space Intersection: Example

Example

Assume $\mathcal{B} = [0,3] \times [0,2]$ and a constraint c: $x + 2 \cdot y \leq 2$.

Contraction for x: $x \leq 2 - 2 \cdot y \Leftrightarrow x \in [0,3] \cap (-\infty, 2] - [0,4] \rightarrow x \in [0,2]$

Contraction for y:
$y \leq (1-x) \div 2 \Leftrightarrow y \in [0,2] \cap ((-\infty, 2] - [0,2]) \div 2 \rightarrow y \in [0,1]$



[2] See [Sch19] for a proof.

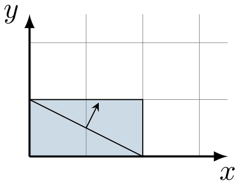
ICP-style Half-space Intersection: Example

Example

Assume $\mathcal{B} = [0,3] \times [0,2]$ and a constraint c: $x + 2 \cdot y \leq 2$.

Contraction for x: $x \leq 2 - 2 \cdot y \Leftrightarrow x \in [0,3] \cap (-\infty, 2] - [0,4] \to x \in [0,2]$

Contraction for y:
$y \leq (1 - x) \div 2 \Leftrightarrow y \in [0,2] \cap ((-\infty, 2] - [0,2]) \div 2 \to y \in [0,1]$



Note: termination not guaranteed due to new intervals.

But: For single linear constraints, a single iteration suffices[2].

[2] See [Sch19] for a proof.

Boxes – operations

Union:

$$\mathcal{B}_c = \mathcal{B}_a \cup \mathcal{B}_b = \{x \mid x \in \mathcal{B}_a \vee x \in \mathcal{B}_b\}$$

Note: The union of two convex sets is not necessarily convex $\rightarrow$ we use the closure (cl) of the union.

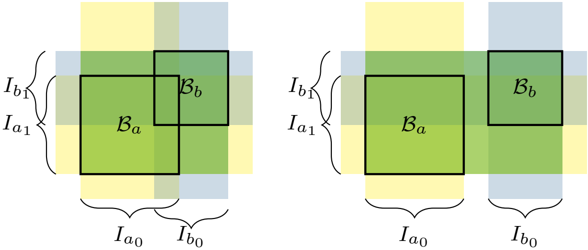
Boxes – operations

Union:

$$\mathcal{B}_c = \mathcal{B}_a \cup \mathcal{B}_b = \{x \mid x \in \mathcal{B}_a \vee x \in \mathcal{B}_b\}$$

Note: The union of two convex sets is not necessarily convex $\rightarrow$ we use the closure (cl) of the union.

$$\begin{aligned}\mathcal{B}_c &= cl(I_{a_0} \cup I_{b_0}), \ldots, cl(I_{a_n} \cup I_{b_n}) \\ &= [\min(I_{a_{0_l}}, I_{b_{0_l}}), \max(I_{a_{0_u}}, I_{b_{0_u}})], \ldots, [\min(I_{a_{n_l}}, I_{b_{n_l}}), \max(I_{a_{n_u}}, I_{b_{n_u}})]\end{aligned}$$



Boxes – operations

Minkowski-sum:

$$\mathcal{B}_c = \mathcal{B}_a \oplus \mathcal{B}_b = \{x \mid x = x_a + x_b, x_a \in \mathcal{B}_a, x_b \in \mathcal{B}_b\}$$

Note: Minkowski's sum can be applied point-wise on convex sets.

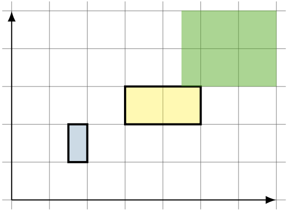
Boxes – operations

Minkowski-sum:

$$\mathcal{B}_c = \mathcal{B}_a \oplus \mathcal{B}_b = \{x \mid x = x_a + x_b, x_a \in \mathcal{B}_a, x_b \in \mathcal{B}_b\}$$

Note: Minkowski's sum can be applied point-wise on convex sets.

$$\begin{aligned}\mathcal{B}_c &= I_{a_0} \oplus I_{b_0}, \dots, I_{a_n} \oplus I_{b_n} \\ &= [I_{a_{0_l}} + I_{b_{0_l}}, I_{a_{0_u}} + I_{b_{0_u}}], \dots, [I_{a_{n_l}} + I_{b_{n_l}}, I_{a_{n_u}} + I_{b_{n_u}}]\end{aligned}$$



Boxes – operations

Linear transformation:

$$\mathcal{B}_c = A \cdot \mathcal{B}_a = \{x \mid x = A \cdot x_a, x_a \in \mathcal{B}_a\}, A \in \mathbb{R}^{n \times n}$$

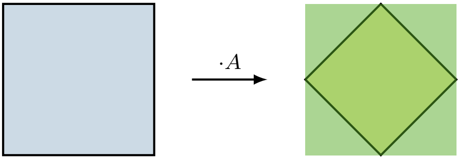
Boxes – operations

Linear transformation:

$$\mathcal{B}_c = A \cdot \mathcal{B}_a = \{x \mid x = A \cdot x_a, x_a \in \mathcal{B}_a\}, A \in \mathbb{R}^{n \times n}$$

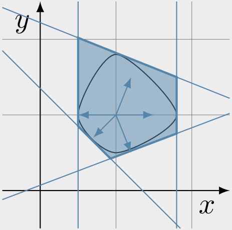
Approaches:

- Naive (conversion): apply A on all vertices, re-convert to box
- Utilize interval arithmetic



Support functions

Definition: support function

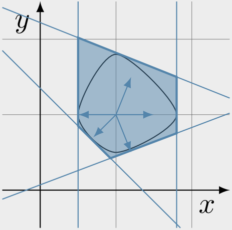


The support function ρ_Ω of a n-dimensional set $\Omega \in \mathbb{R}^n$ is defined as

$$\rho_\Omega : \mathbb{R}^n \to \mathbb{R} \cup \{-\infty, \infty\}$$
$$\rho_\Omega(l) = \sup_{x \in \Omega} \; l^T \cdot x$$

Support functions

Definition: support function



The support function ρ_Ω of a n-dimensional set $\Omega \in \mathbb{R}^n$ is defined as

$$\rho_\Omega : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{-\infty, \infty\}$$
$$\rho_\Omega(l) = \sup_{x \in \Omega} \; l^T \cdot x$$

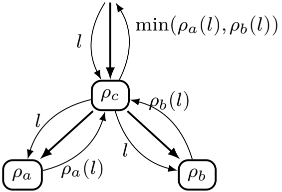
Properties:

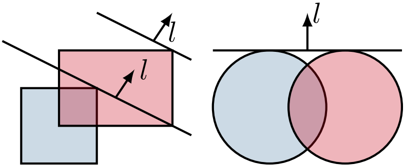
- implemented as tree structure (see next slides)
- operations are cheap, reduced overhead
- scale well in higher dimensions
- well developed (see e.g. [LGG10, FKL13, FGD+11, LG09])

Support functions – operations [LGG10]

Most commonly used operations during reachability analysis:

- Intersection: $\rho_c(l) = \min(\rho_a(l), \rho_b(l))$



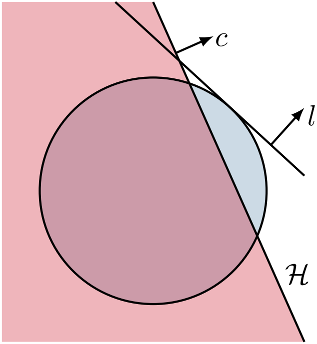


Support functions – operations [LGG10]

Most commonly used operations during reachability analysis:

- Intersection with a half-space $\mathcal{H} = c^T \cdot x \leq d$ (e.g. guards, invariants): $\rho_c(l) = \min(\rho_a(l), \mathcal{H}(l))$,

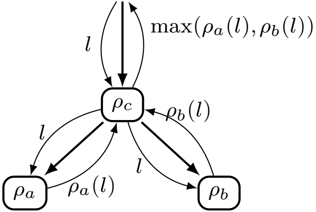
  where $\mathcal{H}(l) = \begin{cases} d & \text{when } l = c \\ \infty & \text{else} \end{cases}$

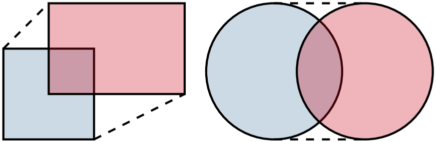


Support functions – operations [LGG10]

Most commonly used operations during reachability analysis:

- Union: $\rho_c(l) = \max(\rho_a(l), \rho_b(l))$



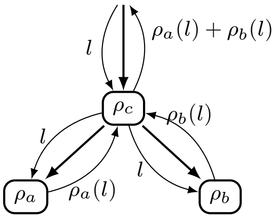


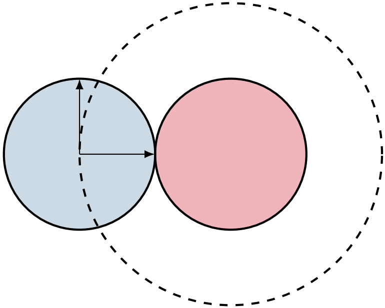
Note: The union operation on a set of support functions returns the supporting hyperplane of the convex hull of the set of underlying sets.

Support functions – operations [LGG10]

Most commonly used operations during reachability analysis:

- Minkowski-sum: $\rho_c(l) = \rho_a(l) + \rho_b(l)$

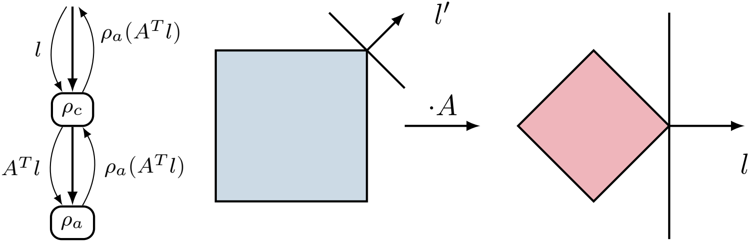




Support functions – operations [LGG10]

Most commonly used operations during reachability analysis:

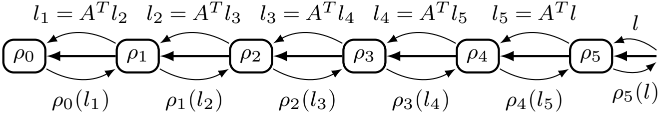
- Linear transformation: $\rho_c = \rho_a(\underbrace{A^T l}_{l'})$



Support functions – optimization

The tree structure in combination with our domain-specific knowledge allows for several optimizations:

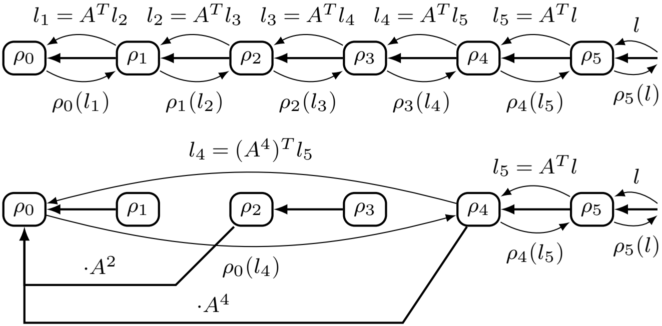
- collect sequences of linear transformations



Support functions – optimization

The tree structure in combination with our domain-specific knowledge allows for several optimizations:

- collect sequences of linear transformations



Support functions – optimization

The tree structure in combination with our domain-specific knowledge allows for several optimizations:

- collect sequences of linear transformations
- remove intersections which have no effect

Support functions – optimization

The tree structure in combination with our domain-specific knowledge allows for several optimizations:

- collect sequences of linear transformations
- remove intersections which have no effect
- reduce tree upon discrete jump (templated evaluation)

Demo

Thermostat[1]

We model and analyze a thermostat according to the following specifications:

- Can either be *on* (initially) or *off*
- Temperature x changes accordingly: $\dot{x} = 50 - x$ (on), $\dot{x} = 10 - x$ (off)
- Switches from on to off when $x \in [20, 25]$
- Switches off to on when $x \in [16, 18]$

[1] https://www.digitalcity.wien/even-thermostats-have-a-heart

Applications

Extensions for reachability analysis based on HyPro:

- Syntactic decoupling - subspace computations
- CEGAR-based reachability analysis

CEGAR-based reachability analysis and parallelization

Parameters for reachability analysis

- Time step size δ
- State set representation
- Aggregation
- ...

CEGAR-based reachability analysis and parallelization

Parameters for reachability analysis

- Time step size δ
- State set representation
- Aggregation
- $\dots$

Reachability analysis induces a search tree, however

- not all branches intersect with bad states $\rightarrow$ coarse analysis
- avoid spurious counterexamples $\rightarrow$ fine analysis

CEGAR-based reachability analysis and parallelization

Parameters for reachability analysis

- Time step size δ
- State set representation
- Aggregation
- ...

Reachability analysis induces a search tree, however

- not all branches intersect with bad states $\rightarrow$ coarse analysis
- avoid spurious counterexamples $\rightarrow$ fine analysis

Goal: Be as lazy as possible and as precise as necessary.

CEGAR-based reachability analysis and parallelization

Goal: Be as lazy as possible and as precise as necessary.

A *parameter setting* collects a full set of relevant parameters, i.e.:

- State set representation R_i
- Time step size δ_i

CEGAR-based reachability analysis and parallelization

Goal: Be as lazy as possible and as precise as necessary.

A *parameter setting* collects a full set of relevant parameters, i.e.:

- State set representation R_i
- Time step size δ_i

Strategy (ordered set of parameter settings):

$$\text{start} \longrightarrow \boxed{R_0,\ \delta_0} \longrightarrow \boxed{R_1,\ \delta_1} \longrightarrow \boxed{R_2,\ \delta_2}$$

CEGAR-based reachability analysis and parallelization

Goal: Be as lazy as possible and as precise as necessary.

A *parameter setting* collects a full set of relevant parameters, i.e.:

- State set representation R_i
- Time step size δ_i

Strategy (ordered set of parameter settings):

start $\rightarrow$ R_0, δ_0 $\rightarrow$ R_1, δ_1 $\rightarrow$ R_2, δ_2

Depending on the application, order and choice of parameter settings matters!

CEGAR-based reachability analysis - Example

Strategy:

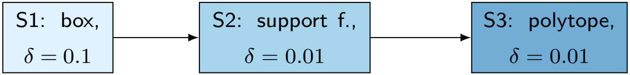
S1: box, $\delta = 0.1$ → S2: support f., $\delta = 0.01$ → S3: polytope, $\delta = 0.01$

Search tree:

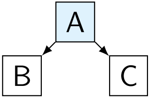
$\boxed{\text{A}}$

CEGAR-based reachability analysis - Example

Strategy:



Search tree:

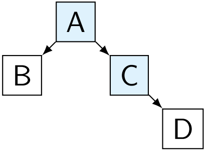


CEGAR-based reachability analysis - Example

Strategy:

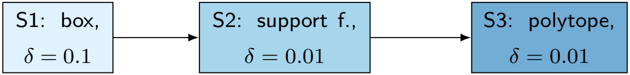
S1: box, $\delta = 0.1$ → S2: support f., $\delta = 0.01$ → S3: polytope, $\delta = 0.01$

Search tree:

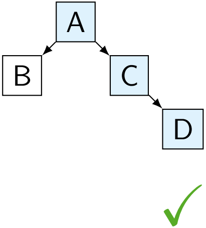


CEGAR-based reachability analysis - Example

Strategy:

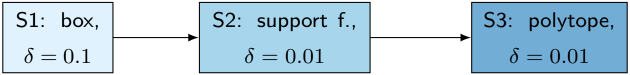


Search tree:

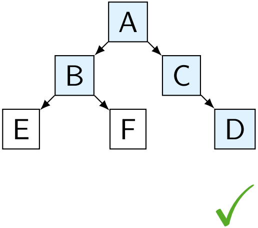


CEGAR-based reachability analysis - Example

Strategy:

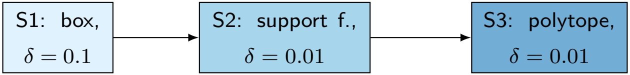


Search tree:

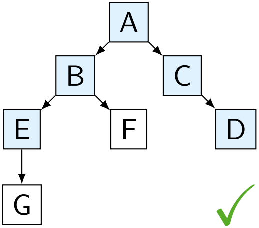


CEGAR-based reachability analysis - Example

Strategy:



Search tree:

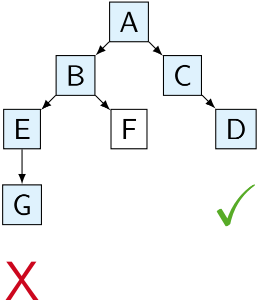


CEGAR-based reachability analysis - Example

Strategy:

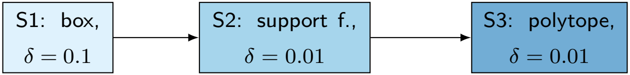
S1: box, $\delta = 0.1$ → S2: support f., $\delta = 0.01$ → S3: polytope, $\delta = 0.01$

Search tree:

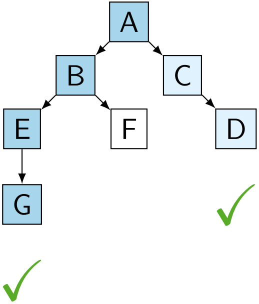


CEGAR-based reachability analysis - Example

Strategy:

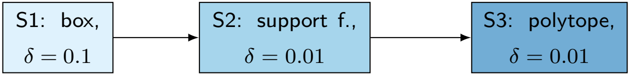


Search tree:

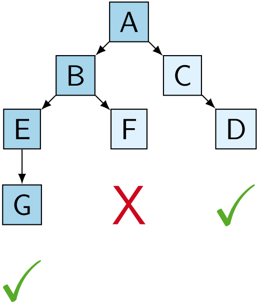


CEGAR-based reachability analysis - Example

Strategy:

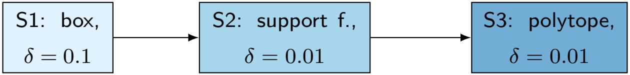


Search tree:

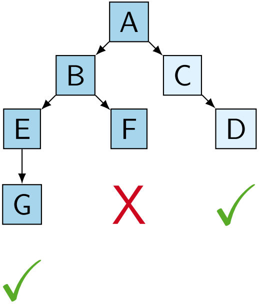


CEGAR-based reachability analysis - Example

Strategy:

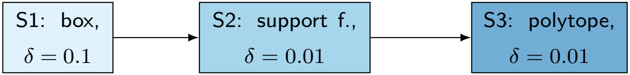


Search tree:

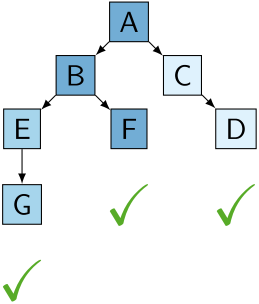


CEGAR-based reachability analysis - Example

Strategy:

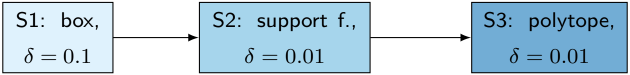


Search tree:

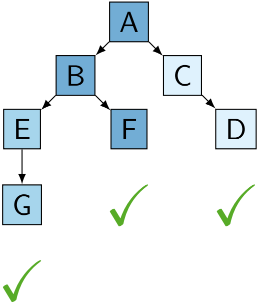


CEGAR-based reachability analysis - Example

Strategy:

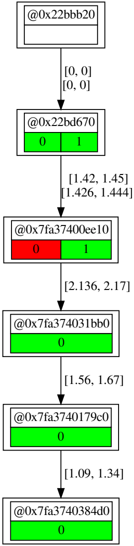


Search tree:

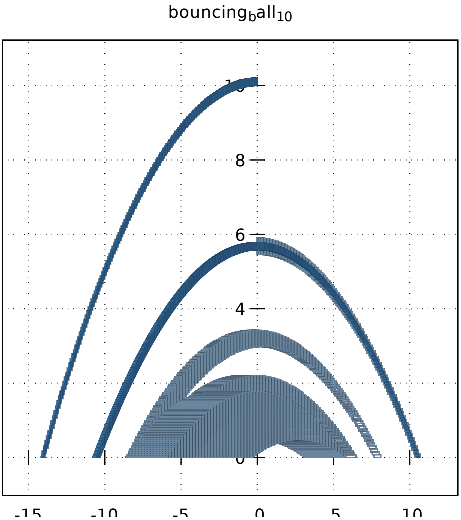


Extension: Parallelized search in different branches.

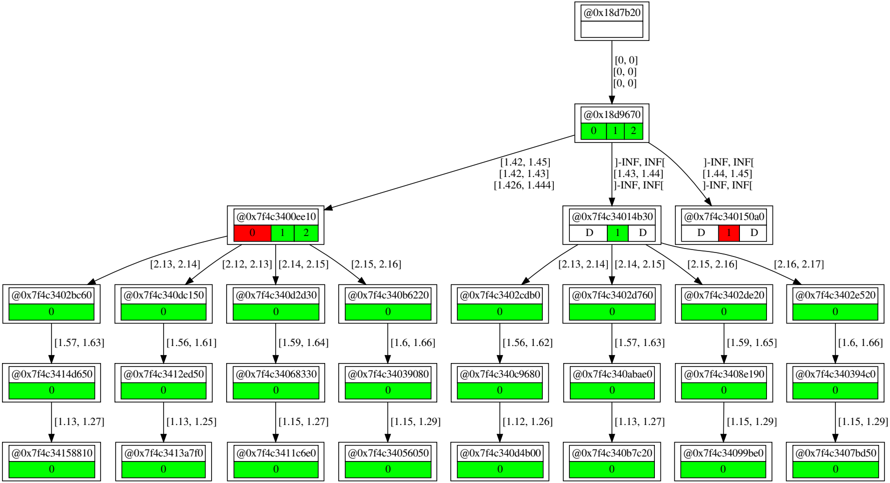
Example: Bouncing ball



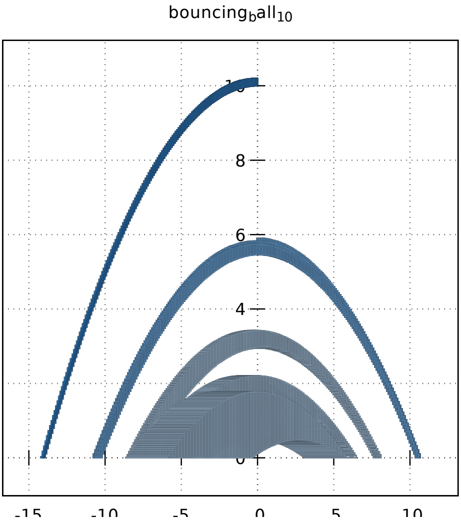
Example: Bouncing ball



Example: Bouncing ball



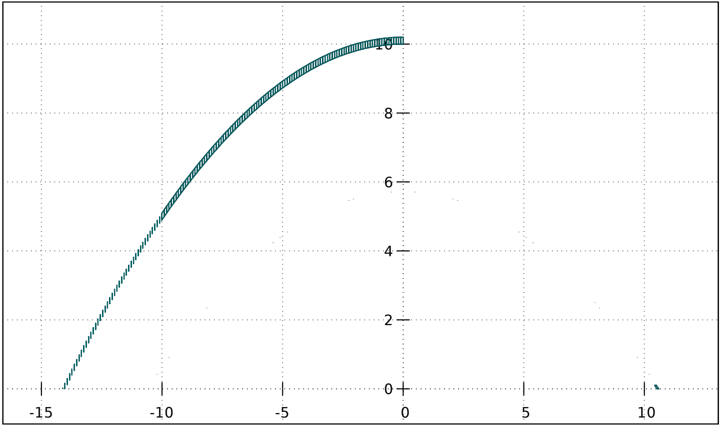
Example: Bouncing ball



A free and open source library for hybrid systems reachability analysis

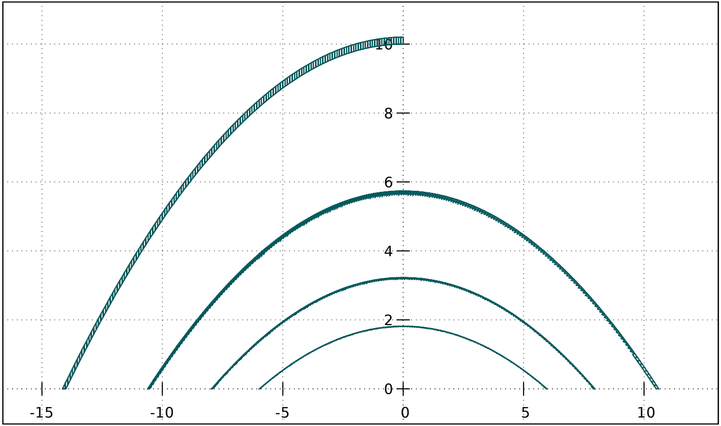
https://github.com/hypro/hypro

Examples



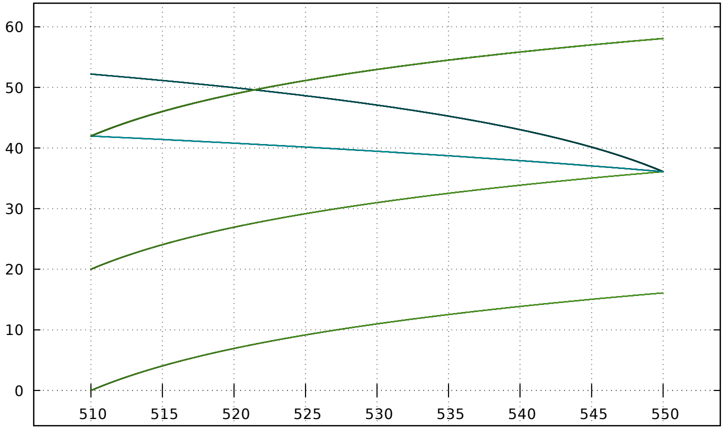
Bouncing ball, $\mathcal{V}$-polytopes with conversion to $\mathcal{H}$-polytopes for intersection, double glpk-only, $T = 3$, $\delta = 0.01$, 4 jumps

Examples



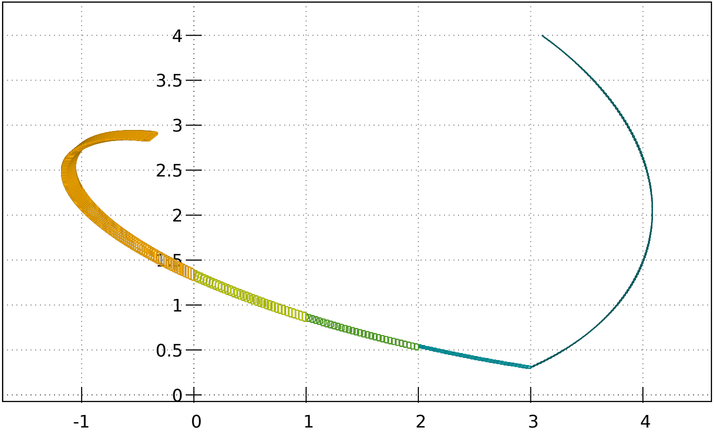
Bouncing ball, $\mathcal{V}$-polytopes with conversion to $\mathcal{H}$-polytopes for intersection, double glpk+SMT-RAT, $T = 3$, $\delta = 0.01$, 4 jumps

Examples



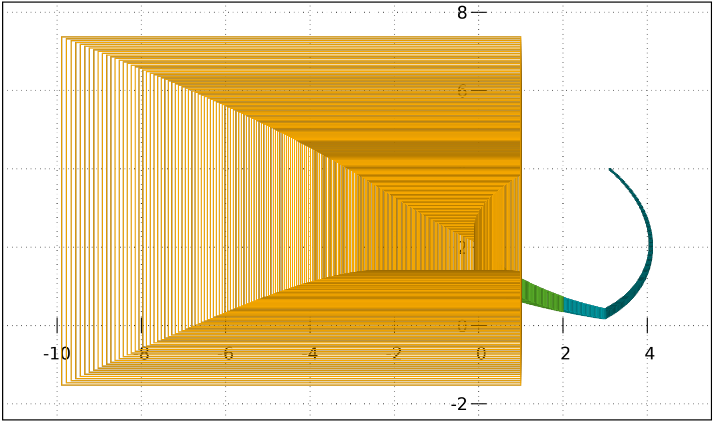
Rod reactor, box, double glpk-only, $T = 17$, $\delta = 0.01$, 2 jumps

Examples



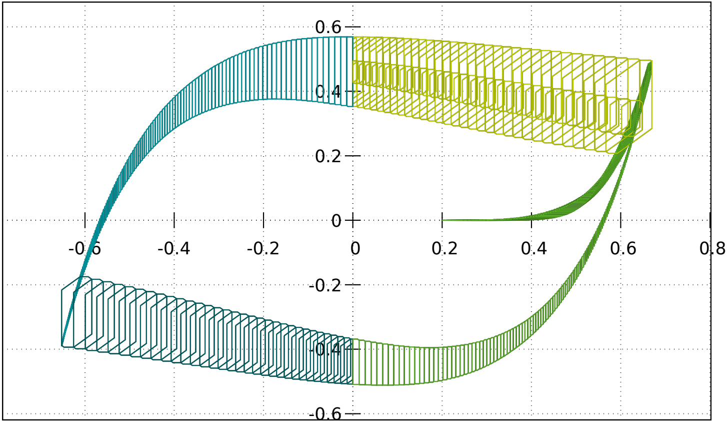
5-D switching system, support function, double glpk-only, $T = 0.2$, $\delta = 0.001$, 4 jumps

Examples



5-D switching system, boxes, double glpk-only, $T = 0.2$, $\delta = 0.001$, 4 jumps

Examples



Filtered oscillator, support function, double glpk-only, $T = 4$, $\delta = 0.01$, 5 jumps