

Set Representations

Luca Geretti

University of Verona, Italy



Outline



- 1 An overview of set representations
- 2 Grid paving set
- 3 Polynomial set
- 4 Combining representations
- 5 Hands-on

Operations expected from a set



■ Essential:

- ▶ **Nonlinear transformation:** required by both continuous and discrete evolution steps
- ▶ **Evaluation / Projection:** necessary for observing the state
- ▶ **Test for membership:** we need to check if a point is in the reachable set

Operations expected from a set



■ Essential:

- ▶ **Nonlinear transformation**: required by both continuous and discrete evolution steps
- ▶ **Evaluation / Projection**: necessary for observing the state
- ▶ **Test for membership**: we need to check if a point is in the reachable set

■ Very important:

- ▶ **Union**: subsets need to be aggregated in order to reason on the complete reachable set in a scalable way
- ▶ **Intersection**: useful for guard activations
- ▶ **Difference**: necessary to understand when new points are reached, for infinite-time reachability

Operations expected from a set



■ Essential:

- ▶ **Nonlinear transformation**: required by both continuous and discrete evolution steps
- ▶ **Evaluation / Projection**: necessary for observing the state
- ▶ **Test for membership**: we need to check if a point is in the reachable set

■ Very important:

- ▶ **Union**: subsets need to be aggregated in order to reason on the complete reachable set in a scalable way
- ▶ **Intersection**: useful for guard activations
- ▶ **Difference**: necessary to understand when new points are reached, for infinite-time reachability

■ Still quite relevant:

- ▶ **Splitting**: since a large set is difficult to handle numerically, we might want to separate it into pieces
- ▶ **Test for emptiness**: it's useful to know efficiently when the set is to be discarded
- ▶ **Bounding box evaluation**: more specific than generic evaluation, this is useful for many methods and it should be as tight as possible

Closure is desirable for operations



- Some set representations **might not be closed** under specific operations.
- It is usually possible to convert into an **enclosing** set in the same or a **different** representation, but at the expense of an **overapproximation error** of varying degree
- When closure is not guaranteed and conversion is undesired, either a **grouping** (specifically a "cover") of sets or a **symbolic storage** of the operation with its operands are possible
 - ▶ Both approaches are meant to **preserve representation** when the cost of conversion is unaffordable
 - ▶ In general, these solutions are **not good for scalability**
 - ▶ In particular cases, however, such composite set **might be simplified** along evolution

Some representations in the literature



■ Geometric objects:

- ▶ boxes (hyper-rectangles) [Moore et al., 2009]
- ▶ oriented rectangular hulls [Stursberg et al., 2003]
- ▶ convex polyhedra [Ziegler, 1995]
- ▶ template polyhedra [Sankaranarayanan et al., 2008]
- ▶ orthogonal polyhedra [Bournez et al., 1999]
- ▶ zonotopes [Girard, 2005])
- ▶ ellipsoids [Kurzanski et al., 2000]

Some representations in the literature



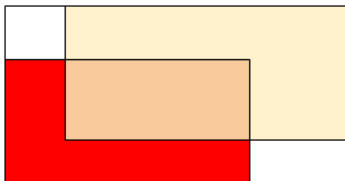
■ Geometric objects:

- ▶ boxes (hyper-rectangles) [Moore et al., 2009]
- ▶ oriented rectangular hulls [Stursberg et al., 2003]
- ▶ convex polyhedra [Ziegler, 1995]
- ▶ template polyhedra [Sankaranarayanan et al., 2008]
- ▶ orthogonal polyhedra [Bournez et al., 1999]
- ▶ zonotopes [Girard, 2005]
- ▶ ellipsoids [Kurzanski et al., 2000]

■ Other (symbolic) representations:

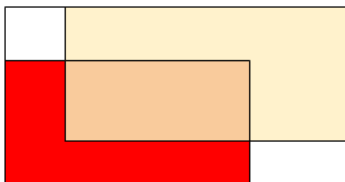
- ▶ support functions [Le Guernic et al., 2009]
- ▶ Taylor models [Berz and Makino, 1998]

Examples

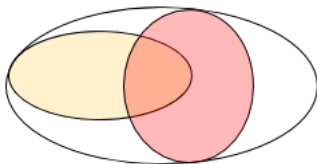


- Intersection is closed
- Difference is not
- Union is easy to perform

Examples



- Intersection is closed
- Difference is not
- Union is easy to perform



- Intersection and difference are not closed
- Union is not that trivial

Switching between representations might be required



- **Accurate** representations are useful for frequent events (such as **continuous steps of evolution**), in order to limit accumulation of overapproximation error;

Switching between representations might be required



- **Accurate** representations are useful for frequent events (such as **continuous steps of evolution**), in order to limit accumulation of overapproximation error;
- **Coarse** representations are useful for sporadic events, where operations such as **intersection, joining and splitting** are required and would be inefficient/ineffective on accurate representations.

In the following



We will provide details regarding two specific set representations:

1. **Grid paving set** (or simply "grid set"): it represents a union of boxes with particular limits in their sizes that allows for efficient operations.

In the following



We will provide details regarding two specific set representations:

1. **Grid paving set** (or simply "grid set"): it represents a union of boxes with particular limits in their sizes that allows for efficient operations.
2. **Polynomial enclosure set** (or simply "Polynomial set"): it symbolically represents a set by a vector of polynomial functions in a chosen basis; we will focus on the Taylor basis, from which a Taylor set (i.e., a vector of Taylor models) is derived.

In the following



We will provide details regarding two specific set representations:

1. **Grid paving set** (or simply "grid set"): it represents a union of boxes with particular limits in their sizes that allows for efficient operations.
2. **Polynomial enclosure set** (or simply "Polynomial set"): it symbolically represents a set by a vector of polynomial functions in a chosen basis; we will focus on the Taylor basis, from which a Taylor set (i.e., a vector of Taylor models) is derived.

A grid set and a Taylor set are at opposite ends of the spectrum:

1. A collection of boxes with additional constraints in the sizes;
2. A generic vector of polynomials with no limit in the accuracy.

VS

Outline



- 1 An overview of set representations
- 2 Grid paving set
- 3 Polynomial set
- 4 Combining representations
- 5 Hands-on

Definitions



Definition (Grid)

A coordinate-aligned discrete partitioning of the variables space, which identifies **cells** of different sizes.

Definition (Grid paving set)

A marking of cells locked to a grid.

Definitions

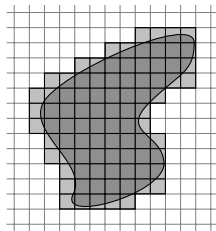
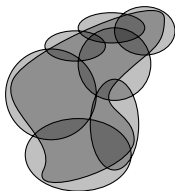
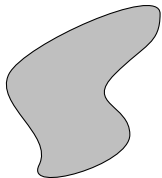


Definition (Grid)

A coordinate-aligned discrete partitioning of the variables space, which identifies **cells** of different sizes.

Definition (Grid paving set)

A marking of cells locked to a grid.



Identify cells in a compact way



Referring to each cell can be done symbolically by exploiting the tiling of the state space that results from the grid. We also are not restricted to using cells of the same size.

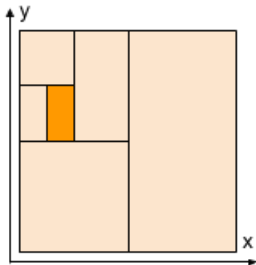
In particular, if we identify a "root cell" with reference center and widths, we can refer to a specific sub-cell by a **binary word** that represents a path downwards from the root:

- Each bit identifies one of the two halves of the given cell in a given dimension;
- The dimension is given by the position of the bit within the word;
- Which half and which dimension is a fixed convention, not encoded
 - ▶ Typically 1 is for the upper half, and the dimensions are visited in order;

Graphical example of a grid cell



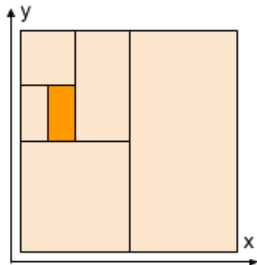
Given a 2-dimensional space (x,y) with a chosen root cell, this means taking the left half of the root cell on x , then the top half on y , again the left half on x , the bottom half on y and the right half on x .



Graphical example of a grid cell



Given a 2-dimensional space (x,y) with a chosen root cell, this means taking the left half of the root cell on x , then the top half on y , again the left half on x , the bottom half on y and the right half on x .

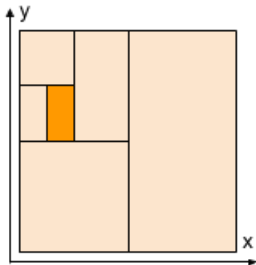


- Since the order of the dimensions is fixed, not all cells can be represented: rectangles wider on x are excluded

Graphical example of a grid cell



Given a 2-dimensional space (x,y) with a chosen root cell, this means taking the left half of the root cell on x , then the top half on y , again the left half on x , the bottom half on y and the right half on x .

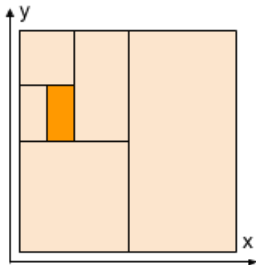


- Since the order of the dimensions is fixed, not all cells can be represented: rectangles wider on x are excluded
- A shorter word of the same prefix identifies a cell containing this one

Graphical example of a grid cell



Given a 2-dimensional space (x,y) with a chosen root cell, this means taking the left half of the root cell on x , then the top half on y , again the left half on x , the bottom half on y and the right half on x .

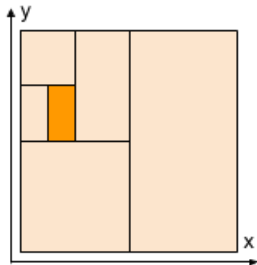


- Since the order of the dimensions is fixed, not all cells can be represented: rectangles wider on x are excluded
- A shorter word of the same prefix identifies a cell containing this one
- Words of the same length identify cells of the same widths

Graphical example of a grid cell



Given a 2-dimensional space (x,y) with a chosen root cell, this means taking the left half of the root cell on x , then the top half on y , again the left half on x , the bottom half on y and the right half on x .



- Since the order of the dimensions is fixed, not all cells can be represented: rectangles wider on x are excluded
- A shorter word of the same prefix identifies a cell containing this one
- Words of the same length identify cells of the same widths
- Another word with different bit values identifies a disjoint cell

Store a grid set in a compact way



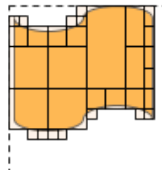
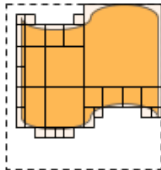
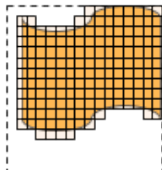
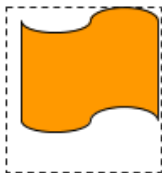
- Since each cell is identified by a binary path from a root, a **binary tree** is the first natural choice
 - ▶ The root node is the root cell
 - ▶ Each node can have zero or two child nodes
 - ▶ Each leaf node can be marked to identify a cell in the set
 - ▶ If two child nodes are marked with the same value, they can be removed and the parent node marked with that value → joining of the cells

Store a grid set in a compact way



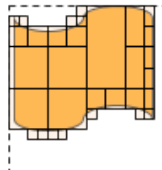
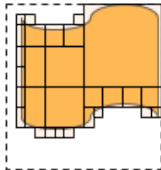
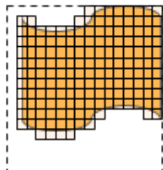
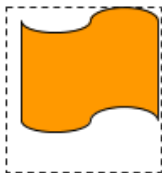
- Since each cell is identified by a binary path from a root, a **binary tree** is the first natural choice
 - ▶ The root node is the root cell
 - ▶ Each node can have zero or two child nodes
 - ▶ Each leaf node can be marked to identify a cell in the set
 - ▶ If two child nodes are marked with the same value, they can be removed and the parent node marked with that value → joining of the cells
- However, a **binary decision diagram** is more efficient at storing the necessary bits.

Example of paving a set with a grid set



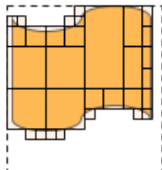
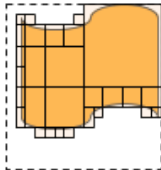
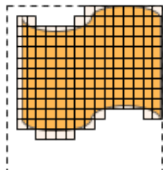
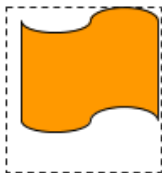
- The grid set in the second figure is useful for splitting a large set into smaller, more numerically amenable, subsets

Example of paving a set with a grid set



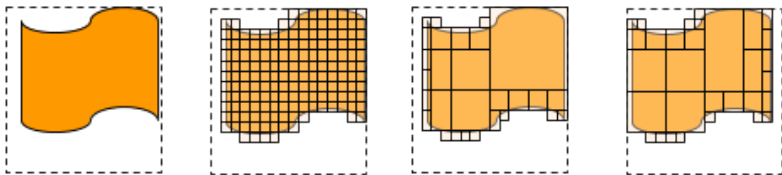
- The grid set in the second figure is useful for splitting a large set into smaller, more numerically amenable, subsets
- The grid set on the third figure is the most efficient when evolution is not considered

Example of paving a set with a grid set



- The grid set in the second figure is useful for splitting a large set into smaller, more numerically amenable, subsets
- The grid set on the third figure is the most efficient when evolution is not considered
- The choice of the root cell (which can be any rectangle centered anywhere) is essential to the efficiency of the grid set approximation

Example of paving a set with a grid set



- The grid set in the second figure is useful for splitting a large set into smaller, more numerically amenable, subsets
- The grid set on the third figure is the most efficient when evolution is not considered
- The choice of the root cell (which can be any rectangle centered anywhere) is essential to the efficiency of the grid set approximation
 - ▶ In the third figure we have 28 cells, in the fourth 31
 - ▶ However, if we want to combine sets, the root cell must be common to all sets in the reachable set

Pros and cons of grid sets



Pros

1. No geometric limitations: it can represent even the union of disjoint non-convex sets;
2. Being based on boxes, it converts easily from/to (a collection of) more complex representations;
3. Allows a compact internal representation;
4. Union, intersection, difference and membership test can be performed very efficiently.

Pros and cons of grid sets



Pros

1. No geometric limitations: it can represent even the union of disjoint non-convex sets;
2. Being based on boxes, it converts easily from/to (a collection of) more complex representations;
3. Allows a compact internal representation;
4. Union, intersection, difference and membership test can be performed very efficiently.

Cons

1. It is still an explicit representation: the number of cells grows exponentially in the number of dimensions.
2. Being its cells aligned to the coordinates, it produces poor approximations when the set is "diagonal" and in particular when it is "long and thin".
3. Being unable to choose the coordinate to split/join into, the number of cells is not always optimal.

Outline



- 1 An overview of set representations
- 2 Grid paving set
- 3 Polynomial set**
- 4 Combining representations
- 5 Hands-on

Symbolic representation



To represent the set we use the image of a function mapping a **parameter space** into the **state space**: $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, i.e., $\{f_j : \mathbb{R}^n \mapsto \mathbb{R}\}_{j=1}^m$.

Symbolic representation



To represent the set we use the image of a function mapping a **parameter space** into the **state space**: $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, i.e., $\{f_j : \mathbb{R}^n \mapsto \mathbb{R}\}_{j=1}^m$.

- For numerical reasons, the domain is the $[-1, 1]^n$ hypercube.

Symbolic representation



To represent the set we use the image of a function mapping a

parameter space into the **state space**: $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, i.e.,

$$\{f_j : \mathbb{R}^n \mapsto \mathbb{R}\}_{j=1}^m.$$

- For numerical reasons, the domain is the $[-1, 1]^n$ hypercube.
- We want $f_j = c_j + \sum_{i=0}^{N_j} a_{ij} \Pi_{ij}(\mathbf{p})$, i.e., a linear combination of products of terms, where each term is chosen in the **polynomial basis** of a parameter.
 - ▶ e.g., if the basis for a generic parameter p_k is $\{1, p_k, p_k^2, p_k^3, \dots\}$, then a valid product Π_{ij} could be $p_1^2 p_3$.

Symbolic representation



To represent the set we use the image of a function mapping a **parameter space** into the **state space**: $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, i.e., $\{f_j : \mathbb{R}^n \mapsto \mathbb{R}\}_{j=1}^m$.

- For numerical reasons, the domain is the $[-1, 1]^n$ hypercube.
- We want $f_j = c_j + \sum_{i=0}^{N_j} a_{ij} \Pi_{ij}(\mathbf{p})$, i.e., a linear combination of products of terms, where each term is chosen in the **polynomial basis** of a parameter.
 - ▶ e.g., if the basis for a generic parameter p_k is $\{1, p_k, p_k^2, p_k^3, \dots\}$, then a valid product Π_{ij} could be $p_1^2 p_3$.
- Coefficients c_j, a_{ij} may be singleton reals or real intervals.

Symbolic representation



To represent the set we use the image of a function mapping a

parameter space into the **state space**: $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, i.e.,

$$\{f_j : \mathbb{R}^n \mapsto \mathbb{R}\}_{j=1}^m.$$

- For numerical reasons, the domain is the $[-1, 1]^n$ hypercube.
- We want $f_j = c_j + \sum_{i=0}^{N_j} a_{ij} \Pi_{ij}(\mathbf{p})$, i.e., a linear combination of products of terms, where each term is chosen in the **polynomial basis** of a parameter.
 - ▶ e.g., if the basis for a generic parameter p_k is $\{1, p_k, p_k^2, p_k^3, \dots\}$, then a valid product Π_{ij} could be $p_1^2 p_3$.
- Coefficients c_j , a_{ij} may be singleton reals or real intervals.
- Since we bound the accuracy, and since each operation on the set must be correctly rounded for overapproximation, we also introduce a **uniform error** term $\pm e$.
 - ▶ Everything not symbolically represented is "swept" into the error term.
 - ▶ See the uniform error as the remainder of a polynomial expansion.

Storing the set



Given that the basis is fixed for all parameters, we only need to store the coefficients and the index in the base for each multiplicand. Summarising, we need:

1. an array of reals or real intervals;
2. an array of multi-index naturals;
3. the real modulus of the error term;

for each dimension of the continuous state space.

Manipulating the set



While the operations of interest in the evolution of a set are done efficiently using Interval Analysis, most other operations are problematic:

- Union is rather coarse;
- Difference is not computable;
- Splitting can be done, but only on the parameter space: this yields overlapping sets in general;
- Testing for membership and intersection are not computable in general;

Manipulating the set



While the operations of interest in the evolution of a set are done efficiently using Interval Analysis, most other operations are problematic:

- Union is rather coarse;
- Difference is not computable;
- Splitting can be done, but only on the parameter space: this yields overlapping sets in general;
- Testing for membership and intersection are not computable in general;

Actually to observe most geometrical properties of the set it is necessary to pass through **range evaluation**

- e.g., by iterative splitting and range evaluation, we can draw an overapproximation of the set boundary

Range evaluation



For $p \in [-1, 1]$, it holds that the range of $ap^2 + bp$ is:

$$a([-1, 1] + b/2a)^2 - b^2/4a$$

Generalised to multiple parameters, the range of $\sum_j a_j p_j^2 + b_j p_j$ is obtained as the sum of the ranges.

- If $a \approx 0$, we use $(|a| + |b|)[-1, 1]$ instead.

Range evaluation



For $p \in [-1, 1]$, it holds that the range of $ap^2 + bp$ is:

$$a([-1, 1] + b/2a)^2 - b^2/4a$$

Generalised to multiple parameters, the range of $\sum_j a_j p_j^2 + b_j p_j$ is obtained as the sum of the ranges.

■ If $a \approx 0$, we use $(|a| + |b|)[-1, 1]$ instead.

Hence, for each dimension whose range we want to evaluate:

1. We initialise the range with the coefficient of the **order 0 term** (i.e., the constant in the polynomial)
2. We collect the coefficients of **order 1 and pure order 2 terms**
3. For each parameter, we use the formula above and add the result to the range
4. All **other terms** are swept into the **uniform error** e
5. We add $[-e, e]$.

Example of a polynomial set



Set $[-1, 1]^2 \mapsto \mathbb{R}^2$ given by

$$x = p_0 + p_1 + p_1^2$$

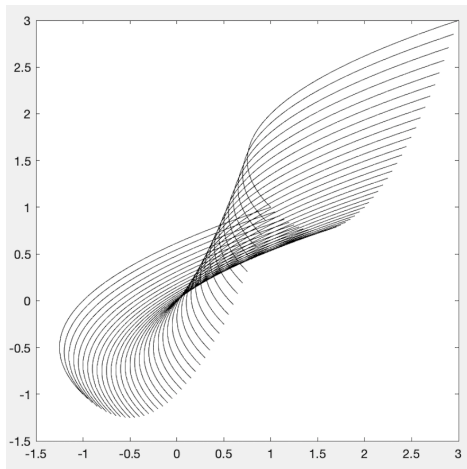
$$y = p_0 + p_1 + p_0^2$$

Evaluation yields for both:

$$[-1, 1] + \left([-1, 1] + \frac{1}{2}\right)^2 - \frac{1}{4} =$$

$$[-1, 1] + \left[-\frac{1}{2}, \frac{3}{2}\right]^2 - \frac{1}{4} =$$

$$[-1, 1] + \left[0, \frac{9}{4}\right] - \frac{1}{4} = \left[-\frac{5}{4}, 3\right]$$



Example of a polynomial set



Set $[-1, 1]^2 \mapsto \mathbb{R}^2$ given by

$$x = p_0 + p_1 + p_1^2$$

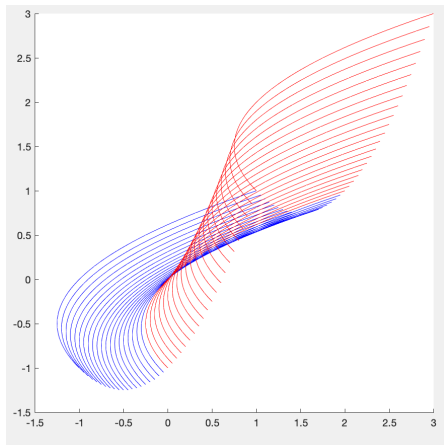
$$y = p_0 + p_1 + p_0^2$$

Evaluation yields for both:

$$[-1, 1] + \left([-1, 1] + \frac{1}{2} \right)^2 - \frac{1}{4} =$$

$$[-1, 1] + \left[-\frac{1}{2}, \frac{3}{2} \right]^2 - \frac{1}{4} =$$

$$[-1, 1] + \left[0, \frac{9}{4} \right] - \frac{1}{4} = \left[-\frac{5}{4}, 3 \right]$$



By splitting along p_0 , i.e. $p'_0 = [-1, 0]$ and $p''_0 = [0, 1]$ we obtain partially overlapping sets.

The choice of the base



- A set is usually created from a function in the first place.
 - ▶ e.g., a constraint, a vector field for the dynamics)
- The degree used for the base is bounded, and the remainder allows to overapproximate the function.
- The base respects the property that the overapproximation error of the expansion goes to zero for degree $n \rightarrow \infty$.
- While the "expanded" representation is polynomial regardless of the bases, different bases at the same degree produce different polynomials.
- The choice of the base affects the number of terms (hence coefficients stored) and the complexity of the operations involved.

Three bases of interest



■ Each has its own generating function, in close form or recursive.

■ The simpler the basis, the easier the algebra.

■ The more complex the basis, the faster the decay of the coefficients (but this still depends on the function to approximate)

- **Taylor** ($B_n = x^n$)

(0) 1

(1) 1, x

(2) 1, x , x^2

(3) 1, x , x^2 , x^3

- **Chebyshev** ($B_n = 2x B_{n-1} - B_{n-2}$)

(0) 1

(1) 1, x

(2) 1, x , $2x^2 - 1$

(3) 1, x , $2x^2 - 1$, $4x^3 - 3x$

- **Bernstein** ($B_{k,n} = \binom{n}{k} x^k (1-x)^{n-k}$)

(0) 1

(1) $1 - x$, x

(2) $(1-x)^2$, $2x(1-x)$, x^2

(3) $(1-x)^3$, $3x(1-x)^2$, $3x^2(1-x)$, x^3

The focus is on the Taylor basis



- In practice the basis of choice for tools dealing with nonlinear dynamics.
 - ▶ The Bernstein basis is used in the tool Sapo.
- Very simple algebra, though potentially the coefficients decay slower with the degree.
 - ▶ Lack of multi-basis tools prevents a solid comparison yet.
- It can be seen as a nonlinear extension of a zonotope.

Controlling the set representation quality



- The higher the degree used, the better the approximation.
 - ▶ However, algebraic operations during evolution could produce a lot of terms in the polynomial representation, even if those have very small coefficients.
- We have two main mechanisms for controlling set quality: enforcing a **bounded accuracy** at all times and performing **reconditioning** periodically.

Bounded accuracy



Allow to decide if a polynomial term should be added into the error e after an algebraic operation (or after the initial construction of the set). We can enforce either or both:

1. maximum polynomial degree;
2. minimum coefficient value.

Bounded accuracy



Allow to decide if a polynomial term should be added into the error e after an algebraic operation (or after the initial construction of the set). We can enforce either or both:

1. maximum polynomial degree;
 2. minimum coefficient value.
- Bounded accuracy causes the error to increase along evolution ("wrapping effect");
 - A large error is numerically bad for any algebraic operation;
 - In particular, if the uniform error is treated as an extra parameter, it can be reduced whenever the set contracts along evolution.

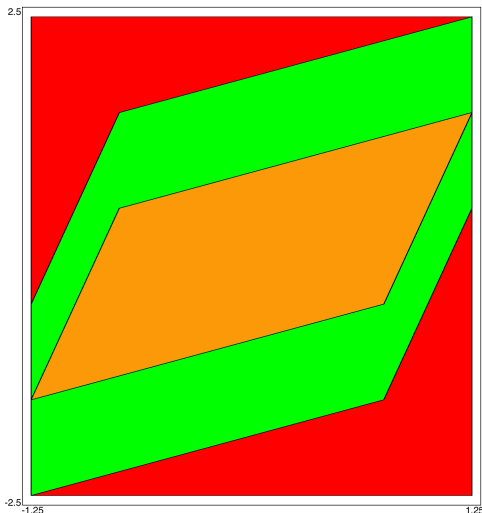
Reconditioning



Trade between accuracy and domain space complexity.

- a. Parametrise: convert e into an additional parameter
→ increase n .
 - ▶ This causes no loss of information.
- b. Sweep: add the absolute value of the coefficients of the terms where a parameter appears into e
→ reduce n .
 - ▶ This is destructive and can introduce an overapproximation.

A linear example on the effects of reconditioning



Original set

$$\begin{aligned}x &= p_0 + 0.25p_1 \pm 0 \\y &= 0.5p_0 + p_1 \pm 0\end{aligned}$$

Add error and convert

$$\begin{aligned}x &= p_0 + 0.25p_1 \pm 0 \\y &= 0.5p_0 + p_1 + p_2 \pm 0 \\ \text{i.e. } y &= 0.5p_0 + p_1 \pm 1\end{aligned}$$

Sweep everything and convert

$$\begin{aligned}x &= 1.25p_0 \pm 0 \\y &= 2.5p_1 \pm 0 \\ \text{i.e. } x &= \pm 1.25, y = \pm 2.5\end{aligned}$$

Pros and cons of polynomial sets



Pros

1. Can be as accurate as desired while being a single vector function;
2. Algebraic operations are very efficient using Interval Arithmetic;

Pros and cons of polynomial sets



Pros

1. Can be as accurate as desired while being a single vector function;
2. Algebraic operations are very efficient using Interval Arithmetic;

Cons

1. Being symbolic, almost all geometric operations are not computable;
2. To recover some observability, evaluation on a splitting of the set is necessary;
3. Splitting can be performed only on the parameter space, leading to overlapping.

Outline



- 1 An overview of set representations
- 2 Grid paving set
- 3 Polynomial set
- 4 Combining representations**
- 5 Hands-on

Grid sets vs polynomial sets



- They have opposite geometric properties in terms of accuracy and computability.
- We should use grid sets only when absolutely necessary, when computability demands it, in order to avoid the conversion error into cells.

Grid sets vs polynomial sets



- They have opposite geometric properties in terms of accuracy and computability.
- We should use grid sets only when absolutely necessary, when computability demands it, in order to avoid the conversion error into cells.

When is it absolutely necessary?

As we will see, **infinite-time reachability** requires to compute the set of new points reached, until no new points are found. This in turn requires the **set difference** operation, which can not be computed between polynomial sets.

Improvements over this approach?



- Arguably, set representation is the most important aspect of numerical reachability;

Improvements over this approach?



- Arguably, set representation is the most important aspect of numerical reachability;
- No clear-cut choices exist in the nonlinear case, research is still very active;

Improvements over this approach?



- Arguably, set representation is the most important aspect of numerical reachability;
- No clear-cut choices exist in the nonlinear case, research is still very active;
- The use of a secondary representation for set difference (and consequently infinite-time reachability) is almost unexplored territory
 - ▶ Apart from Ariadne, all libraries using the numerical approximation approach focus on the finite-time reachable set

Improvements over this approach?



- Arguably, set representation is the most important aspect of numerical reachability;
- No clear-cut choices exist in the nonlinear case, research is still very active;
- The use of a secondary representation for set difference (and consequently infinite-time reachability) is almost unexplored territory
 - ▶ Apart from Ariadne, all libraries using the numerical approximation approach focus on the finite-time reachable set
- Is a more flexible secondary representation worth the inevitable computation overhead?
 - ▶ Template grid set? Orthogonal polyhedra?

Outline



- 1 An overview of set representations
- 2 Grid paving set
- 3 Polynomial set
- 4 Combining representations
- 5 Hands-on

Try constrained image sets and grid sets



- Constrained image sets are more convenient representations than Taylor sets for plotting and discretising, for these exercises;
- Actual Taylor sets will be displayed when working with continuous evolution.