Efficient Computation of Reachable Sets of Linear Time-Invariant Systems with Inputs

Colas Le Guernic*
École Normale Supérieure

joint work with

Antoine Girard*
University of Pennsylvania

Oded Maler
VERIMAG

March 29, 2006

*currently at VERIMAG

Motivations

- Discrete Linear Time Invariant System:

$$x_{k+1} = \Phi x_k + u_k \quad x_0 \in \Omega_0, \forall i \, u_i \in U$$

Motivations

- Discrete Linear Time Invariant System:

$$x_{k+1} = \Phi x_k + u_k \quad x_0 \in \Omega_0, \forall i \; u_i \in U$$

  - Obtained by discretisation of a continuous system
  - Input can take into account errors due to linearisation and discretisation

Motivations

- Discrete Linear Time Invariant System:

$$x_{k+1} = \Phi x_k + u_k \quad x_0 \in \Omega_0, \forall i\, u_i \in U$$

  - Obtained by discretisation of a continuous system
  - Input can take into account errors due to linearisation and discretisation

- Reachable sets:
  - Set of points reachable from a specified initial set with the considered dynamic under any possible input
  - Computation required for safety verification, controller synthesis,. . .

- Discrete Linear Time Invariant System:

$$x_{k+1} = \Phi x_k + u_k \quad x_0 \in \Omega_0, \forall i\, u_i \in U$$

  - Obtained by discretisation of a continuous system
  - Input can take into account errors due to linearisation and discretisation

- Reachable sets:
  - Set of points reachable from a specified initial set with the considered dynamic under any possible input
  - Computation required for safety verification, controller synthesis,. . .

We will not detail here how Ω_0, Φ and U can be obtained from a continuous time system.

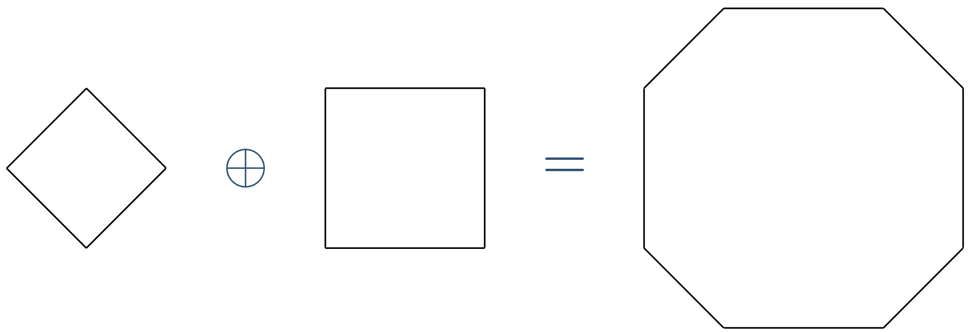
DLTI

We want to compute the N first sets of the sequence defined by:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

- Ω_0 is the set of initial points
- U is the set of inputs
- Φ is a $d \times d$ matrix
- $\oplus$ is the Minkowski sum

$$A \oplus B = \{a + b | a \in A \text{ and } b \in B\}$$



A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

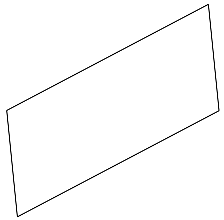
For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.

A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:

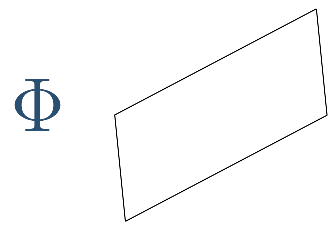


A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:

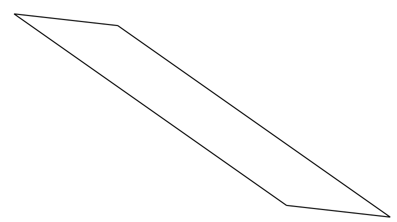


A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:

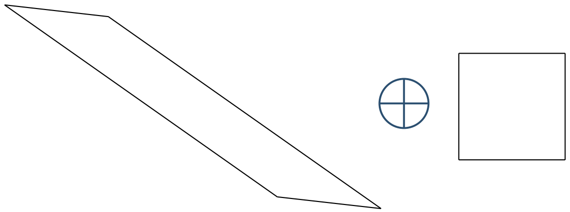


A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:

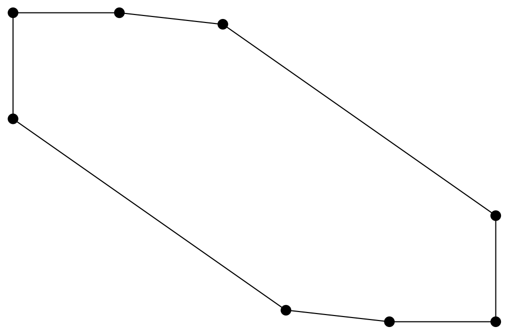


A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi \Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:

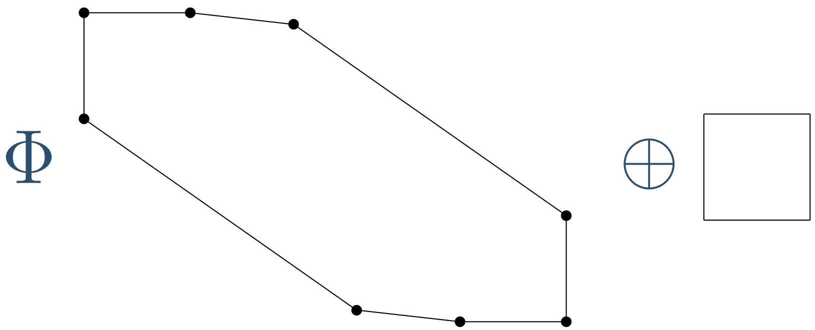


A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:

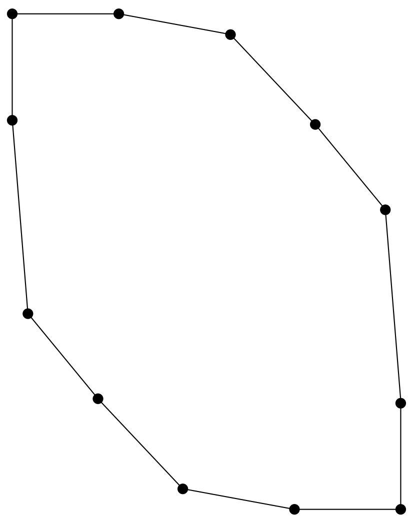


A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:

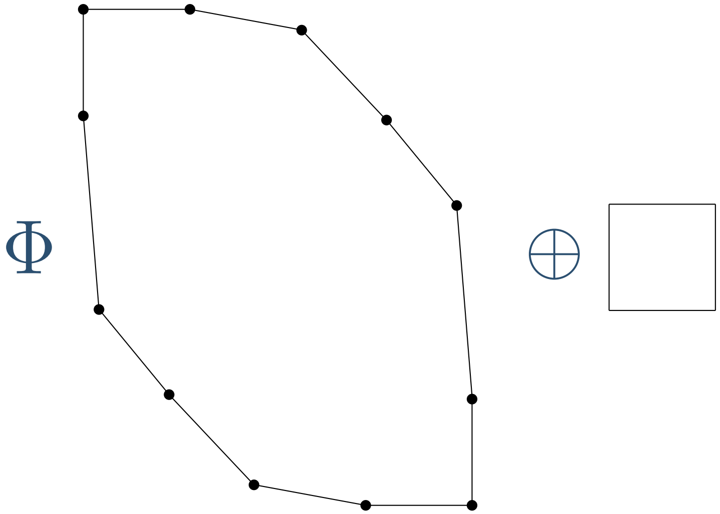


A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

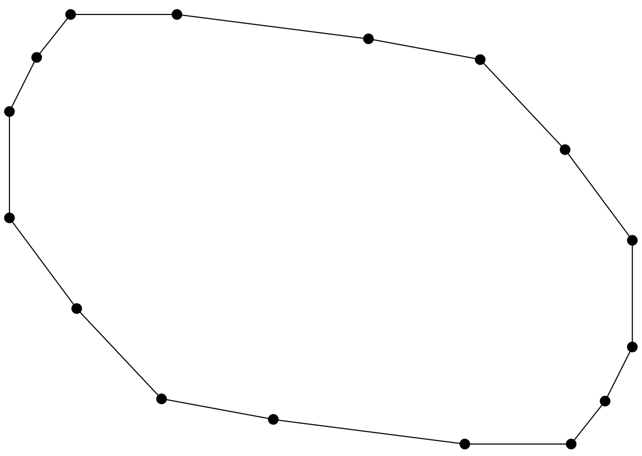
For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:



Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:

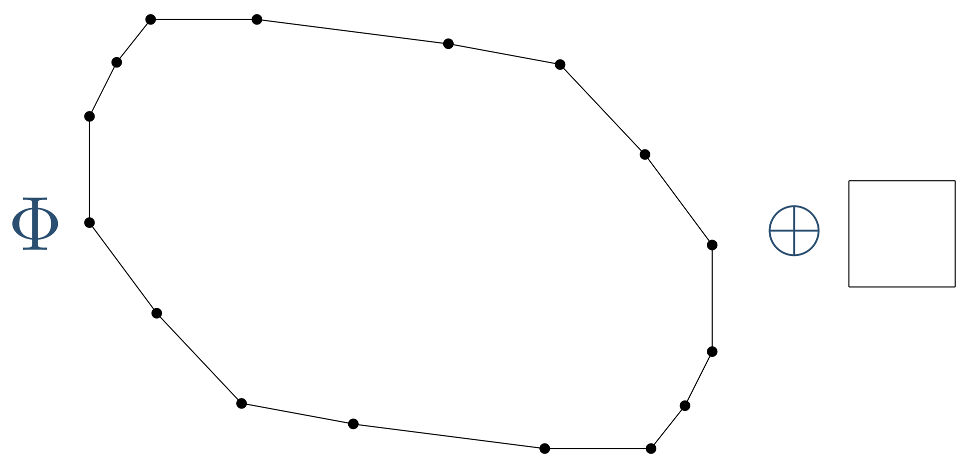


A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:



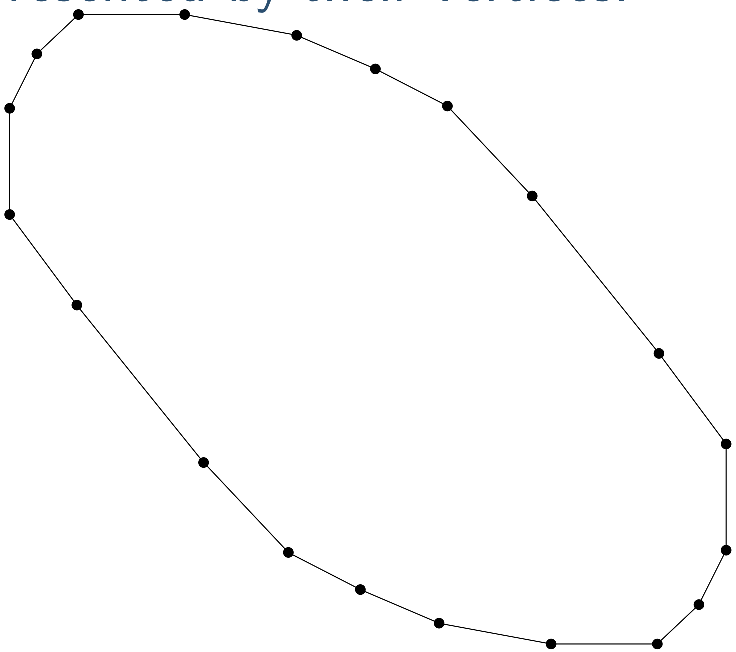
A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.

But:



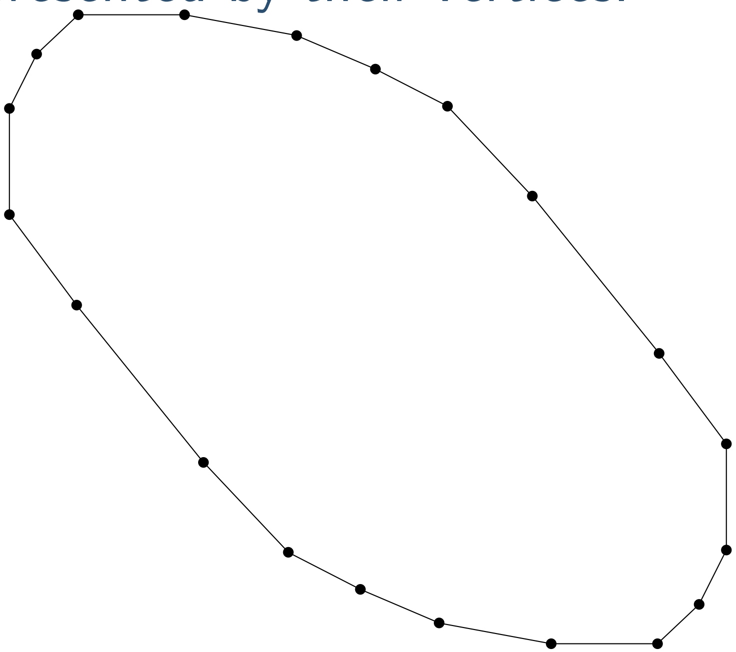
A naive algorithm

Direct use of the recurence relation:

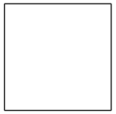
$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:

Φ



$\oplus$



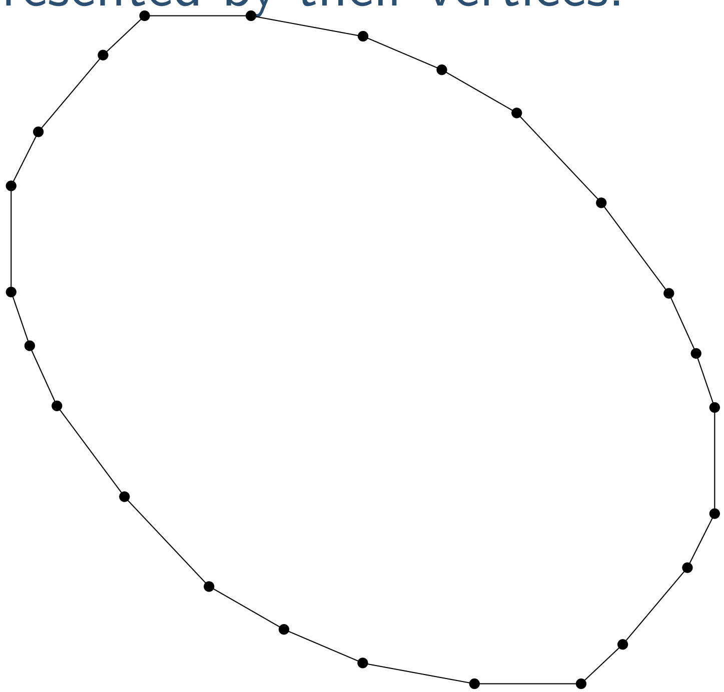
A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.

But:

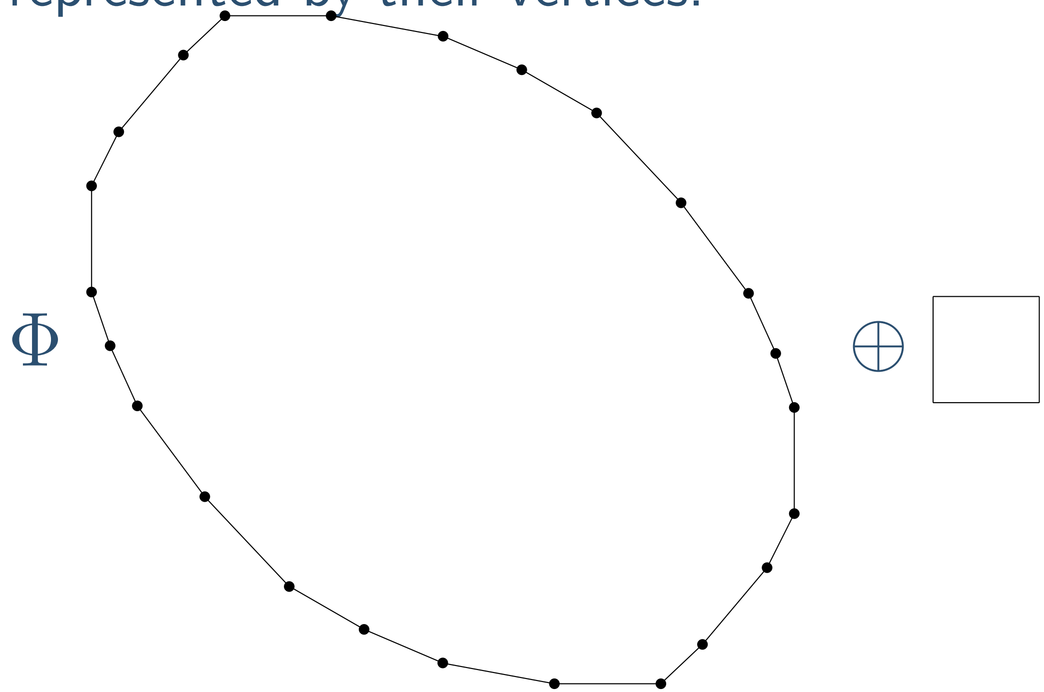


A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:



A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:

$$\ldots$$

A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:

Ω_{n-1} may have more than $\frac{(2n)^{d-1}}{\sqrt{d}}$ vertices.

A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.
But:

Ω_{n-1} may have more than $\frac{(2n)^{d-1}}{\sqrt{d}}$ vertices.

$\Phi\Omega_{n-1}$ needs more than $(2n)^{d-1}d\sqrt{d}$ multiplications.

A naive algorithm

Direct use of the recurence relation:

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

For that, we need a class of sets closed under linear transformation and Minkowski sum, for example: convex polytopes represented by their vertices.

This naive algorithm has complexity about N^{d-1}.

where:

- N is the number of steps considered. ($N \in [100; 1000]$)
- d is the dimension of the system. ($d \in [2; 100]$)

Usual Solution: Approximation

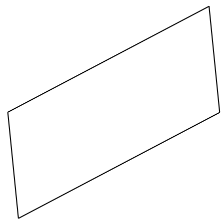
$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.

Usual Solution: Approximation

$$\overline{\Omega}_{n+1} = \text{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

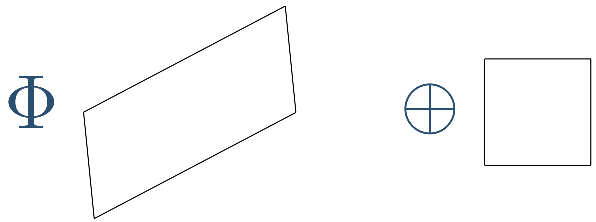
APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.
But:



Usual Solution: Approximation

$$\overline{\Omega}_{n+1} = \text{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

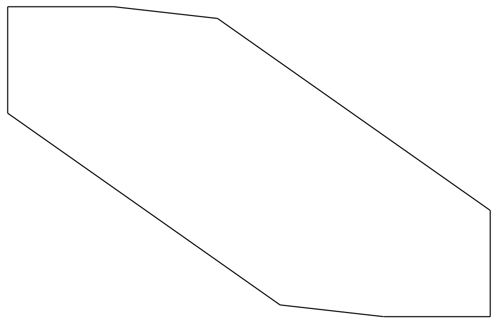
APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.
But:



Usual Solution: Approximation

$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

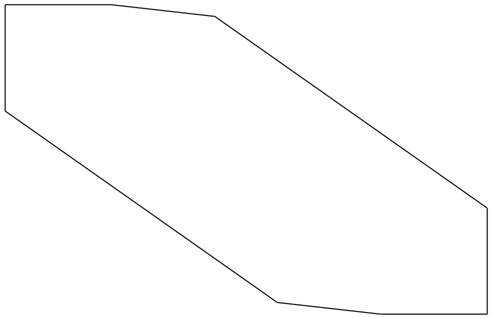
APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.
But:



Usual Solution: Approximation

$$\overline{\Omega}_{n+1} = \text{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

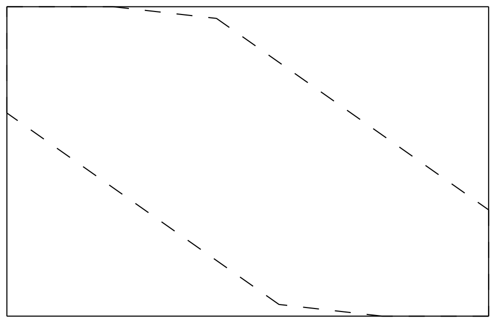
APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.
But:

APPROX()

Usual Solution: Approximation

$$\overline{\Omega}_{n+1} = \text{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

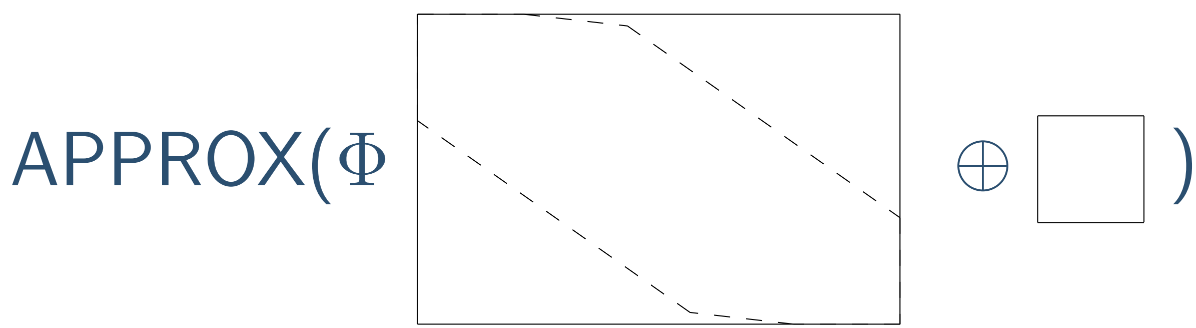
APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.
But:



Usual Solution: Approximation

$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

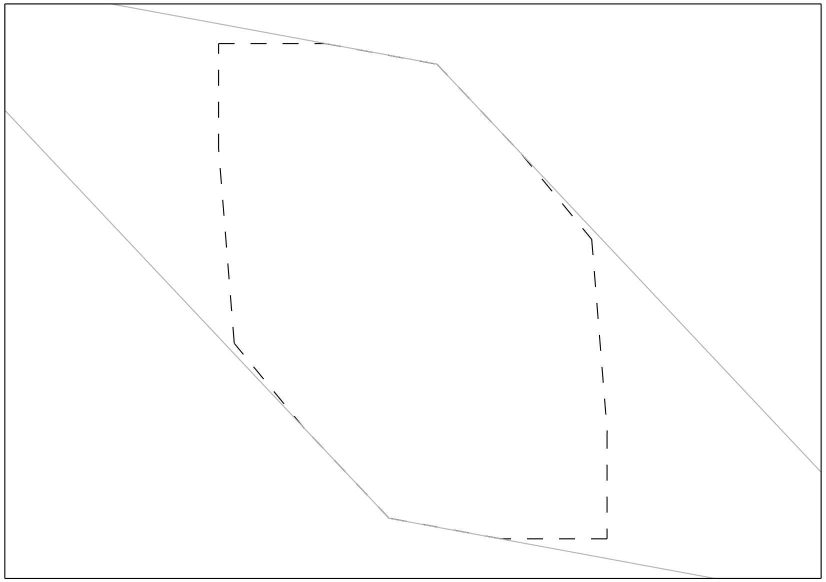
APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.
But:



Usual Solution: Approximation

$$\overline{\Omega}_{n+1} = \text{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

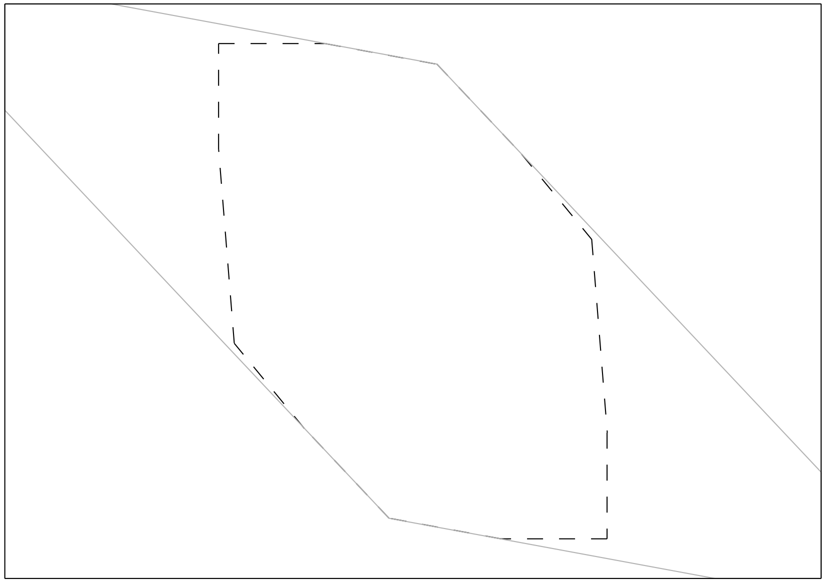
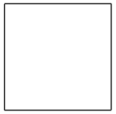
APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.
But:



Usual Solution: Approximation

$$\overline{\Omega}_{n+1} = \text{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.
But:

APPROX(Φ  $\oplus$ )

Usual Solution: Approximation

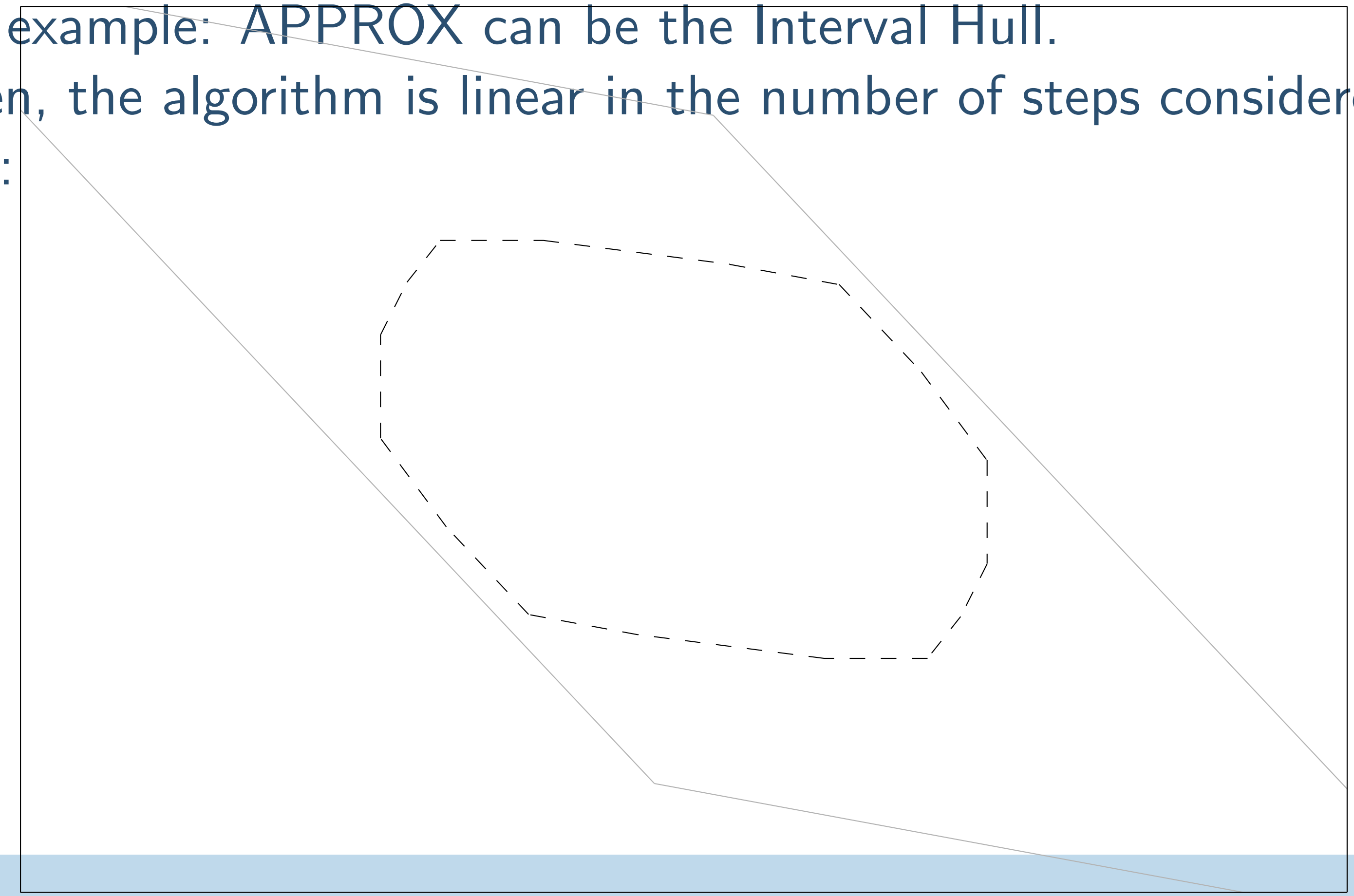
$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

APPROX takes a set and computes an over-approximation with bounded representation size.

For example: APPROX can be the Interval Hull.

Then, the algorithm is linear in the number of steps considered.

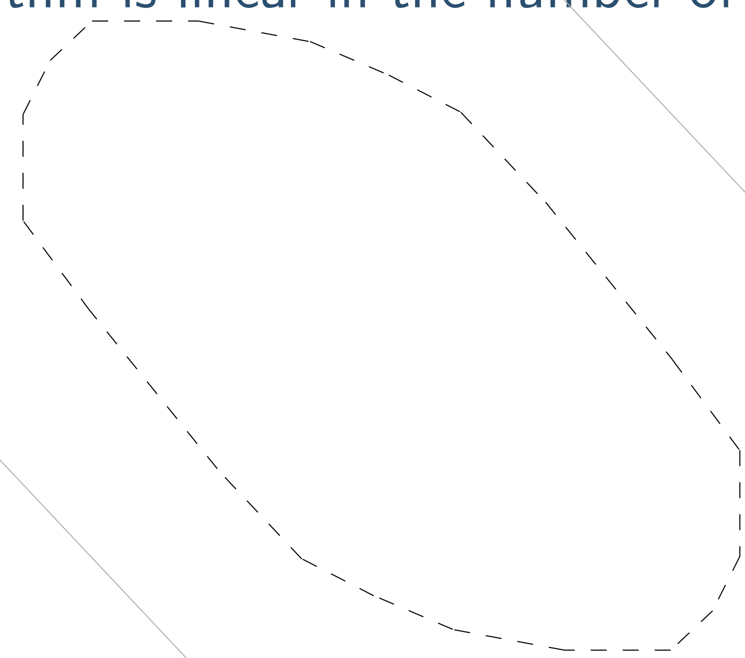
But:



Usual Solution: Approximation

$$\overline{\Omega}_{n+1} = \text{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

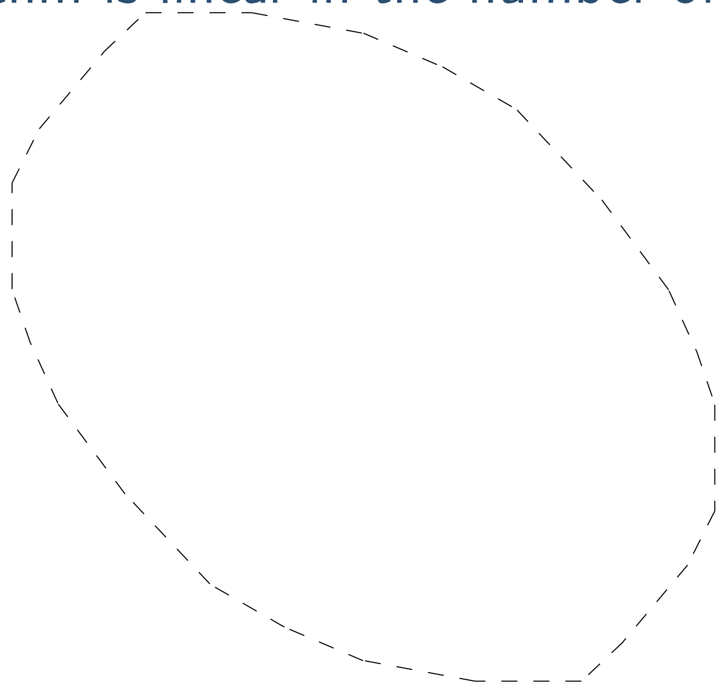
APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.
But:



Usual Solution: Approximation

$$\overline{\Omega}_{n+1} = \text{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.
But:



Usual Solution: Approximation

$$\overline{\Omega}_{n+1} = \text{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.
But:

...

$$\overline{\Omega}_{n+1} = \text{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.
But:

The approximation error can be exponential in the number of steps!

Usual Solution: Approximation

$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

APPROX takes a set and computes an over-approximation with bounded representation size.
For example: APPROX can be the Interval Hull.
Then, the algorithm is linear in the number of steps considered.
But:

The approximation error can be exponential in the number of steps!
Most of the effort has been made on looking for a suitable APPROX function.

Tight approximation

How to evaluate if an APPROX function is suitable?
One that minimizes the volume? the Hausdorff distance?...

Tight approximation

How to evaluate if an APPROX function is suitable?
One that minimizes the volume? the Hausdorff distance?...
These criteria are often hard to evaluate, because they are not conserved by linear transformation.

How to evaluate if an APPROX function is suitable?
An easy to check criterion: Tightness [Kurzhanskiy,Varaiya].
Does the exact set Ω_n “touch” the boundaries of its over-approximation $\overline{\Omega}_n$?

Tight approximation

How to evaluate if an APPROX function is suitable?
An easy to check criterion: Tightness [Kurzhanskiy,Varaiya].
Does the exact set Ω_n “touch” the boundaries of its over-approximation $\overline{\Omega}_n$?
If yes, this contact occurs in a specific direction ℓ_n and (if we deal with convex sets):

$$\max\{x \bullet \ell_n | x \in \Omega_n\} = \max\{x \bullet \ell_n | x \in \overline{\Omega}_n\}$$

Tight approximation

How to evaluate if an APPROX function is suitable?
An easy to check criterion: Tightness [Kurzhanskiy,Varaiya].
Does the exact set Ω_n "touch" the boundaries of its over-approximation $\overline{\Omega}_n$?
If yes, this contact occurs in a specific direction ℓ_n and (if we deal with convex sets):

$$\max\{x \bullet \ell_n | x \in \Omega_n\} = \max\{x \bullet \ell_n | x \in \overline{\Omega}_n\}$$

$$\max\{\Phi^{-1}x \bullet \ell_n | x \in \Phi\Omega_n\} = \max\{\Phi^{-1}x \bullet \ell_n | x \in \Phi\overline{\Omega}_n\}$$

$$\max\{x \bullet (\Phi^{-1})^T\ell_n | x \in \Phi\Omega_n\} = \max\{x \bullet (\Phi^{-1})^T\ell_n | x \in \Phi\overline{\Omega}_n\}$$

Tight approximation

How to evaluate if an APPROX function is suitable?
An easy to check criterion: Tightness [Kurzhanskiy,Varaiya].
Does the exact set Ω_n "touch" the boundaries of its over-approximation $\overline{\Omega}_n$?
If yes, this contact occurs in a specific direction ℓ_n and (if we deal with convex sets):

$$\max\{x \bullet \ell_n | x \in \Omega_n\} = \max\{x \bullet \ell_n | x \in \overline{\Omega}_n\}$$

$$\max\{\Phi^{-1}x \bullet \ell_n | x \in \Phi\Omega_n\} = \max\{\Phi^{-1}x \bullet \ell_n | x \in \Phi\overline{\Omega}_n\}$$

$$\max\{x \bullet (\Phi^{-1})^T\ell_n | x \in \Phi\Omega_n\} = \max\{x \bullet (\Phi^{-1})^T\ell_n | x \in \Phi\overline{\Omega}_n\}$$

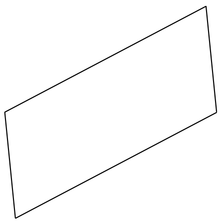
Thus $\ell_{n+1} = (\Phi^{-1})^T\ell_n$, and APPROX only needs to be tight in the direction given by $(\Phi^{-n})^T\ell_0$.

Example

$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

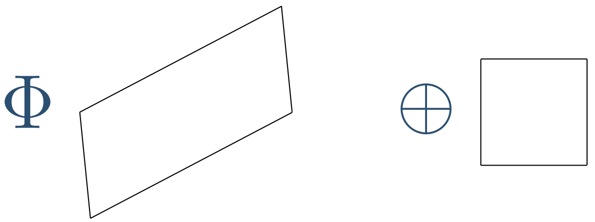
Example

$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$



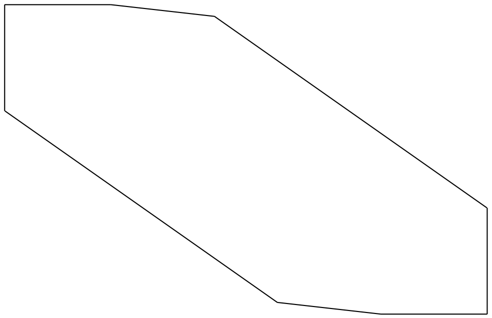
Example

$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$



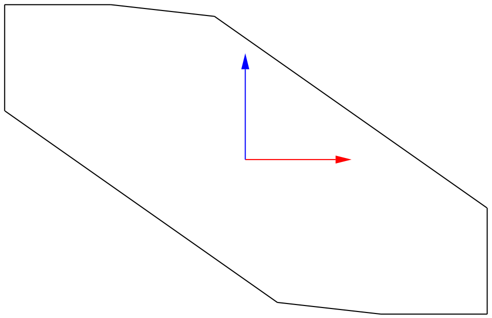
Example

$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$



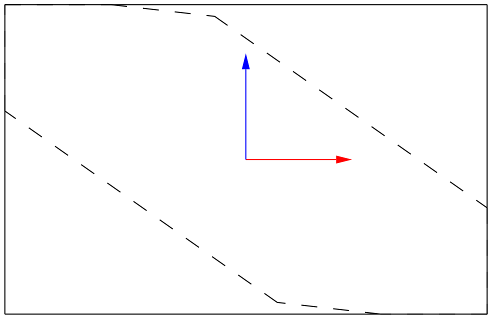
Example

$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

APPROX()

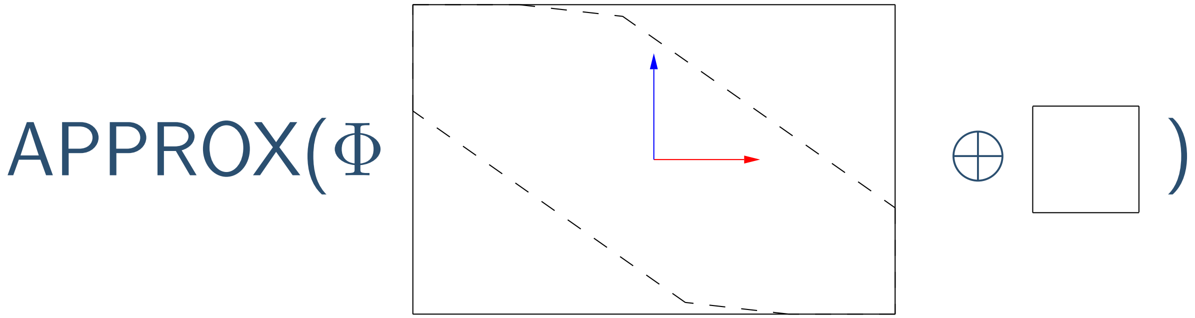
Example

$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$



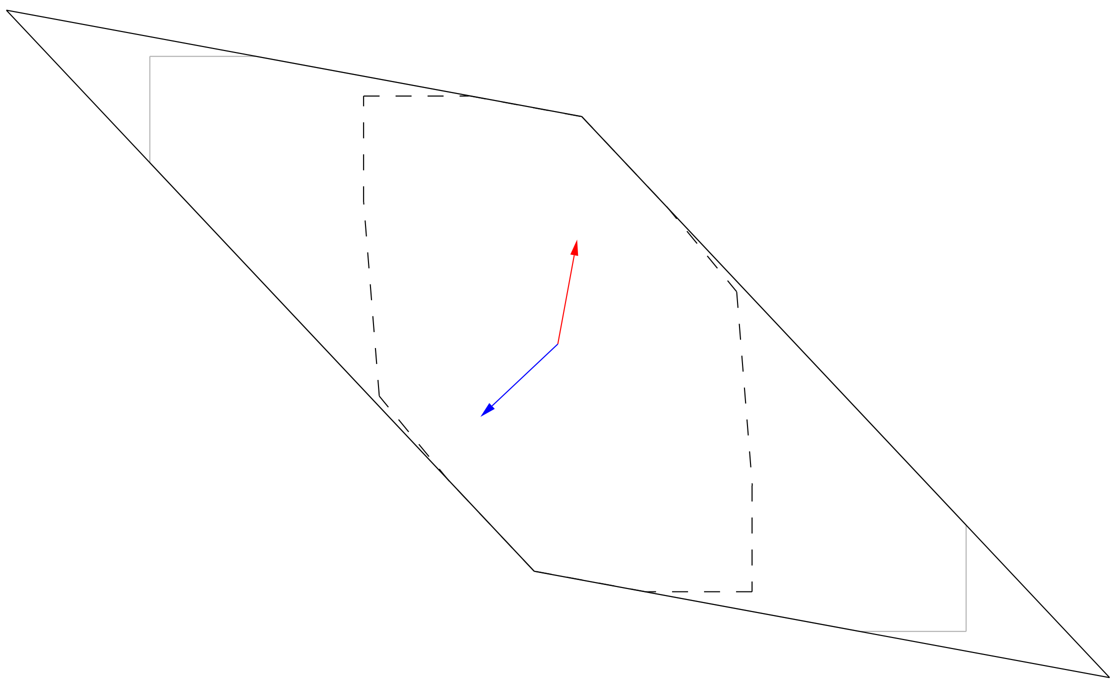
Example

$$\overline{\Omega}_{n+1} = \text{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$



Example

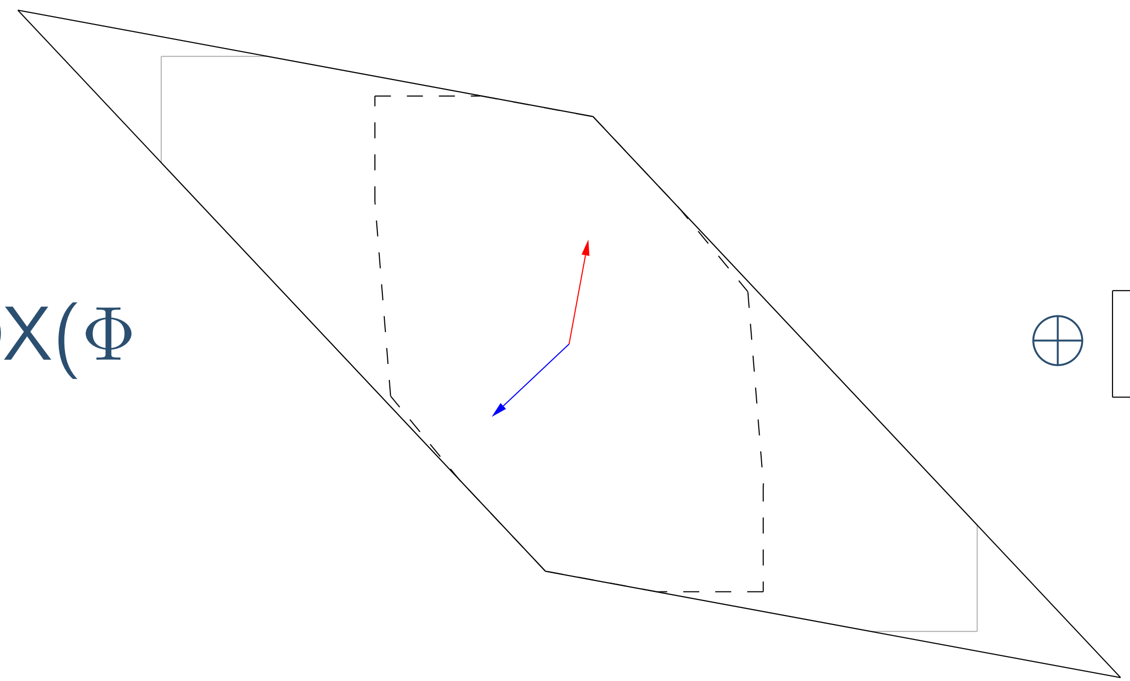
$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$



Example

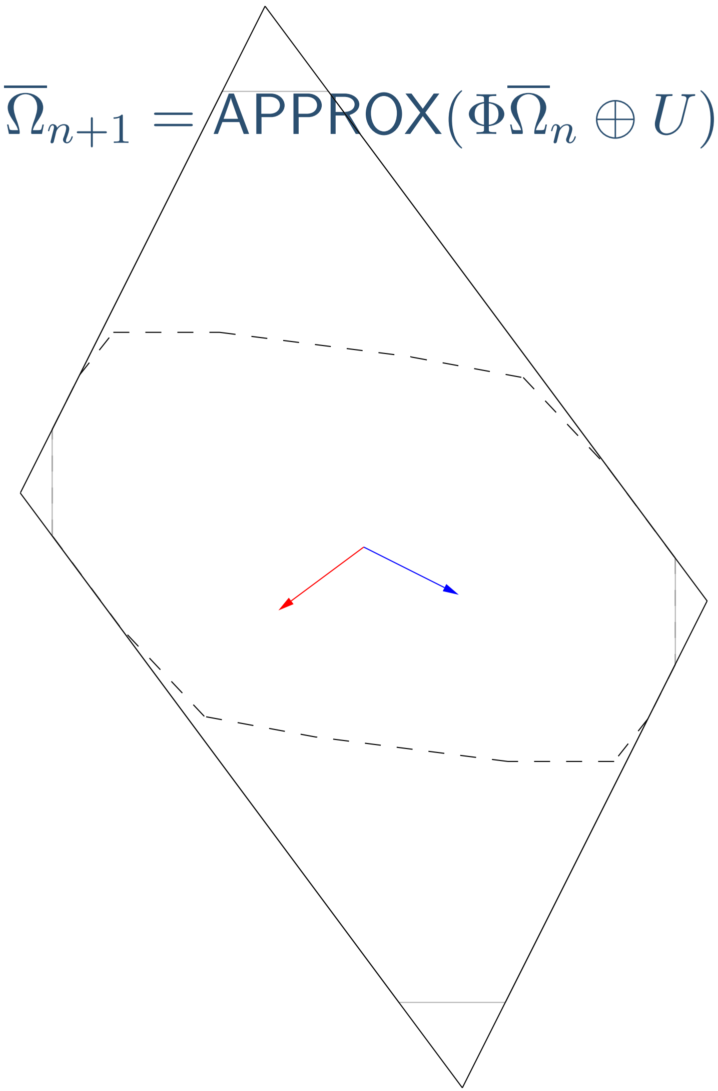
$$\overline{\Omega}_{n+1} = \text{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

$\text{APPROX}(\Phi$



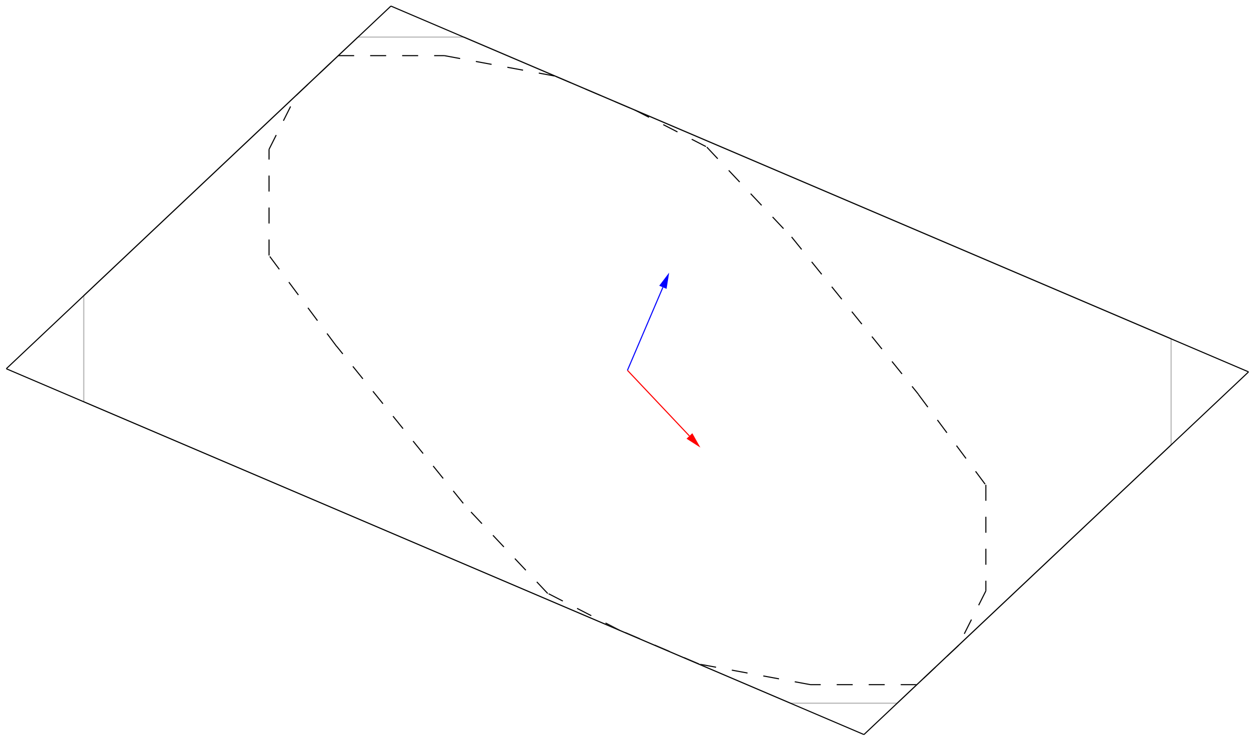
$\oplus \; \square \;)$

$$\overline{\Omega}_{n+1} = \mathrm{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$



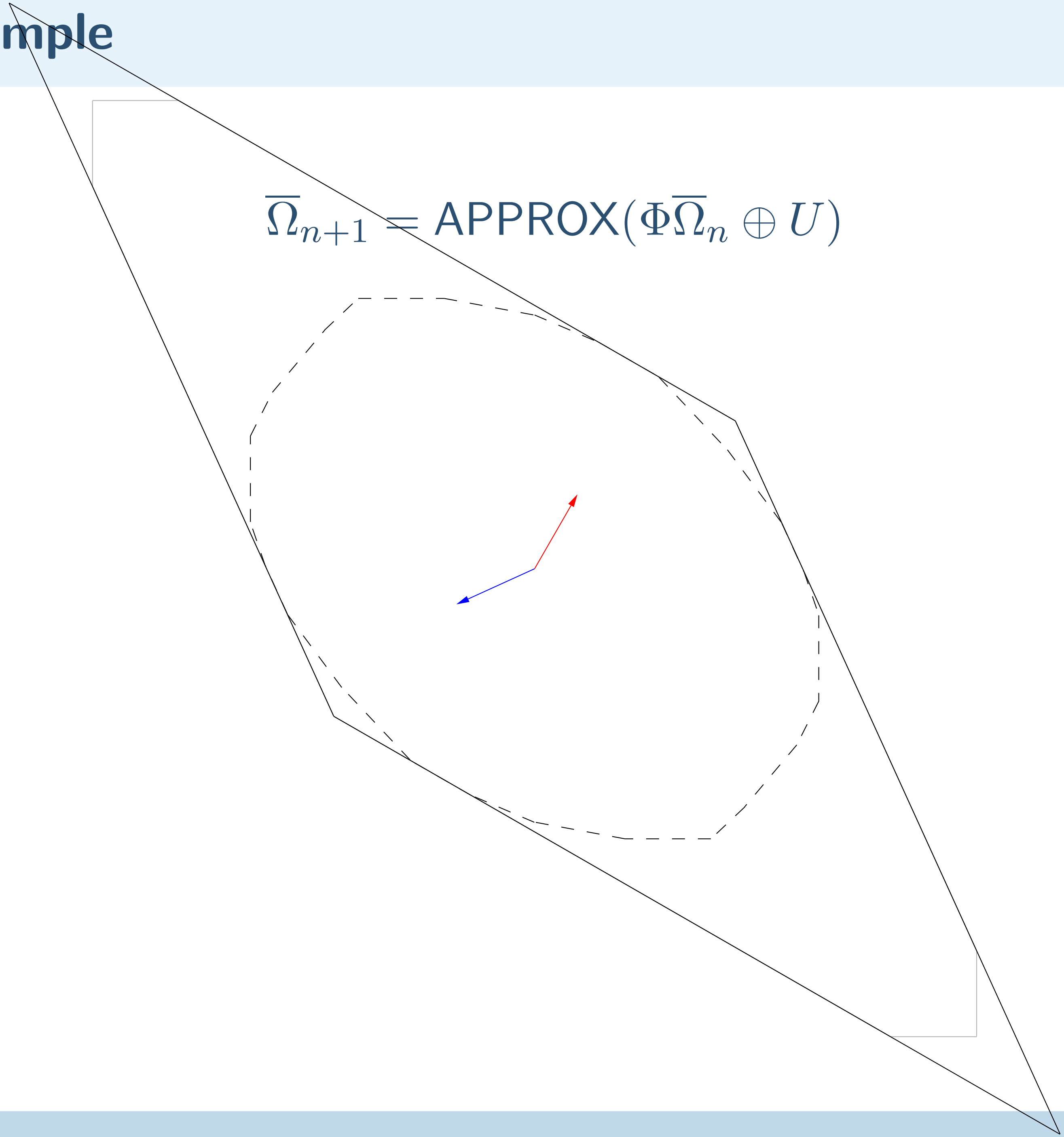
Example

$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$



Example

$$\overline{\Omega}_{n+1} = \text{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$



$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

...

Example

$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

This is much better.

$$\overline{\Omega}_{n+1} = \mathsf{APPROX}(\Phi\overline{\Omega}_n \oplus U)$$

This is much better.

But:

- no reported algorithm has bound on the error in terms of diameter, volume, distance,...
- in some case, all approximation directions may converge toward the same vector.

A new algorithm

A simple idea

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

The problem comes from the mixing of the Minkowski sum (increases the complexity of the considered sets) and linear transformation (propagates the errors).
We should separate these two operations.

A simple idea

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

The problem comes from the mixing of the Minkowski sum (increases the complexity of the considered sets) and linear transformation (propagates the errors).
We should separate these two operations.

$$\Omega_0$$

A simple idea

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

The problem comes from the mixing of the Minkowski sum (increases the complexity of the considered sets) and linear transformation (propagates the errors).
We should separate these two operations.

$$\Omega_1 = \Phi\Omega_0 \oplus U$$

A simple idea

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

The problem comes from the mixing of the Minkowski sum (increases the complexity of the considered sets) and linear transformation (propagates the errors).
We should separate these two operations.

$$\Omega_2 = \Phi(\Phi\Omega_0 \oplus U) \oplus U$$

A simple idea

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

The problem comes from the mixing of the Minkowski sum (increases the complexity of the considered sets) and linear transformation (propagates the errors).
We should separate these two operations.

$$\Omega_2 = \Phi^2\Omega_0 \oplus \Phi U \oplus U$$

A simple idea

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

The problem comes from the mixing of the Minkowski sum (increases the complexity of the considered sets) and linear transformation (propagates the errors).
We should separate these two operations.

$$\Omega_3 = \Phi(\Phi^2\Omega_0 \oplus \Phi U \oplus U) \oplus U$$

A simple idea

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

The problem comes from the mixing of the Minkowski sum (increases the complexity of the considered sets) and linear transformation (propagates the errors).
We should separate these two operations.

$$\Omega_3 = \Phi^3\Omega_0 \oplus \Phi^2 U \oplus \Phi U \oplus U$$

A simple idea

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

The problem comes from the mixing of the Minkowski sum (increases the complexity of the considered sets) and linear transformation (propagates the errors).
We should separate these two operations.

$$\dots$$

A simple idea

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

The problem comes from the mixing of the Minkowski sum (increases the complexity of the considered sets) and linear transformation (propagates the errors).
We should separate these two operations.

$$\Omega_n = \Phi^n\Omega_0 \oplus \bigoplus_{i=0}^{n-1} \Phi^i U$$

A simple idea

$$\Omega_{n+1} = \Phi\Omega_n \oplus U$$

The problem comes from the mixing of the Minkowski sum (increases the complexity of the considered sets) and linear transformation (propagates the errors).
We should separate these two operations.

$$\Omega_n = \Phi^n\Omega_0 \oplus \bigoplus_{i=0}^{n-1} \Phi^i U$$

To compute Ω_n you need two linear transformations (on $\Phi^{n-1}\Omega_0$ and $\Phi^{n-2}U$) and two Minkowski sums.

Exact Algorithm

It is enough to compute the three following sequences:

- $X_0 = \Omega_0$, $X_n = \Phi X_{n-1}$ $\qquad (X_n = \Phi^n \Omega_0)$
- $V_0 = U$, $V_n = \Phi V_{n-1}$ $\qquad (V_n = \Phi^n U)$
- $S_0 = \{0\}$, $S_n = S_{n-1} \oplus V_{n-1}$ $\qquad (S_n = \bigoplus_{i=0}^{n-1} \Phi^i U)$

then $\Omega_n = X_n \oplus S_n$.

Exact Algorithm

It is enough to compute the three following sequences:

- $X_0 = \Omega_0$, $X_n = \Phi X_{n-1}$ $\quad (X_n = \Phi^n \Omega_0)$
- $V_0 = U$, $V_n = \Phi V_{n-1}$ $\quad (V_n = \Phi^n U)$
- $S_0 = \{0\}$, $S_n = S_{n-1} \oplus V_{n-1}$ $\quad (S_n = \bigoplus_{i=0}^{n-1} \Phi^i U)$

then $\Omega_n = X_n \oplus S_n$.
We can now forget about linear transformations (they are performed on constant complexity sets)
We should focus on Minkowski sum:

- we can use Zonotopes [Girard] time complexity is $\mathcal{O}(Nd^3)$, space complexity is $\mathcal{O}(Nd^2)$
  - recall that the naive algorithm with vertices representation has time complexity $\mathcal{O}(N^{d-1})$
- or approximate

Interval Hull Approximation

- $X_0 = \Omega_0,\ X_n = \Phi X_{n-1}$ $\qquad (X_n = \Phi^n \Omega_0)$
- $V_0 = U,\ V_n = \Phi V_{n-1}$ $\qquad (V_n = \Phi^n U)$
- $S_0 = \{0\},\ S_n = S_{n-1} \oplus \mathsf{BOX}(V_{n-1})$ $\qquad (\bigoplus_{i=0}^{n-1} \mathsf{BOX}(\Phi^i U)\ )$

and $\overline{\Omega}_n = \mathsf{BOX}(X_n) \oplus S_n$.

Interval Hull Approximation

- $X_0 = \Omega_0,\ X_n = \Phi X_{n-1}$ $\quad (X_n = \Phi^n \Omega_0)$
- $V_0 = U,\ V_n = \Phi V_{n-1}$ $\quad (V_n = \Phi^n U)$
- $S_0 = \{0\},\ S_n = S_{n-1} \oplus \text{BOX}(V_{n-1})$ $\quad (\bigoplus_{i=0}^{n-1} \text{BOX}(\Phi^i U)\ )$

and $\overline{\Omega}_n = \text{BOX}(X_n) \oplus S_n$.
but for any sets A and B: $\text{BOX}(A) \oplus \text{BOX}(B) = \text{BOX}(A \oplus B)$

Interval Hull Approximation

- $X_0 = \Omega_0$, $X_n = \Phi X_{n-1}$ $\quad (X_n = \Phi^n \Omega_0)$
- $V_0 = U$, $V_n = \Phi V_{n-1}$ $\quad (V_n = \Phi^n U)$
- $S_0 = \{0\}$, $S_n = S_{n-1} \oplus \text{BOX}(V_{n-1})$ $\quad (\text{BOX}(\bigoplus_{i=0}^{n-1} \Phi^i U))$

and $\overline{\Omega}_n = \text{BOX}(X_n) \oplus S_n$.
but for any sets A and B: $\text{BOX}(A) \oplus \text{BOX}(B) = \text{BOX}(A \oplus B)$
thus $\overline{\Omega}_n = \text{BOX}(\Omega_n)$

No wrapping effect!

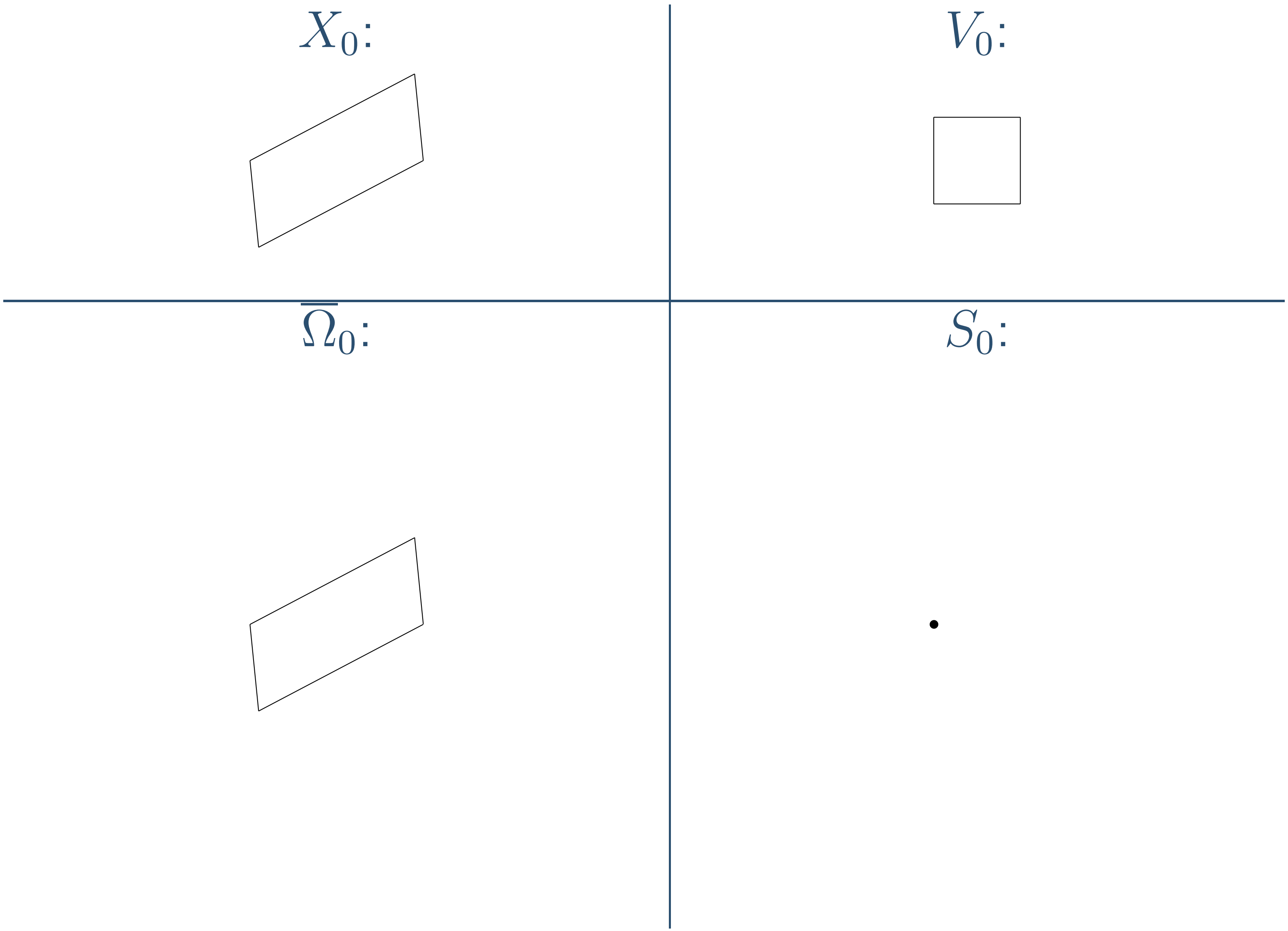
- $X_0 = \Omega_0,\ X_n = \Phi X_{n-1}$ $\quad(X_n = \Phi^n \Omega_0)$
- $V_0 = U,\ V_n = \Phi V_{n-1}$ $\quad(V_n = \Phi^n U)$
- $S_0 = \{0\},\ S_n = S_{n-1} \oplus \text{BOX}(V_{n-1})$ $\quad(\text{BOX}(\bigoplus_{i=0}^{n-1} \Phi^i U))$

and $\overline{\Omega}_n = \text{BOX}(X_n) \oplus S_n$.
but for any sets A and B: $\text{BOX}(A) \oplus \text{BOX}(B) = \text{BOX}(A \oplus B)$
thus $\overline{\Omega}_n = \text{BOX}(\Omega_n)$

No wrapping effect!

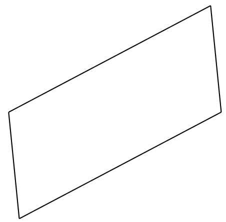
- time complexity: $\mathcal{O}(Nd^3)$ (as the exact algorithm)
- space complexity: $\mathcal{O}(d^2 + Nd)$ (d times smaller)

Example

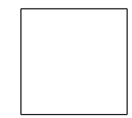


Example

X_1:

Φ 

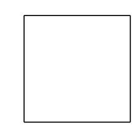
V_1:

Φ 

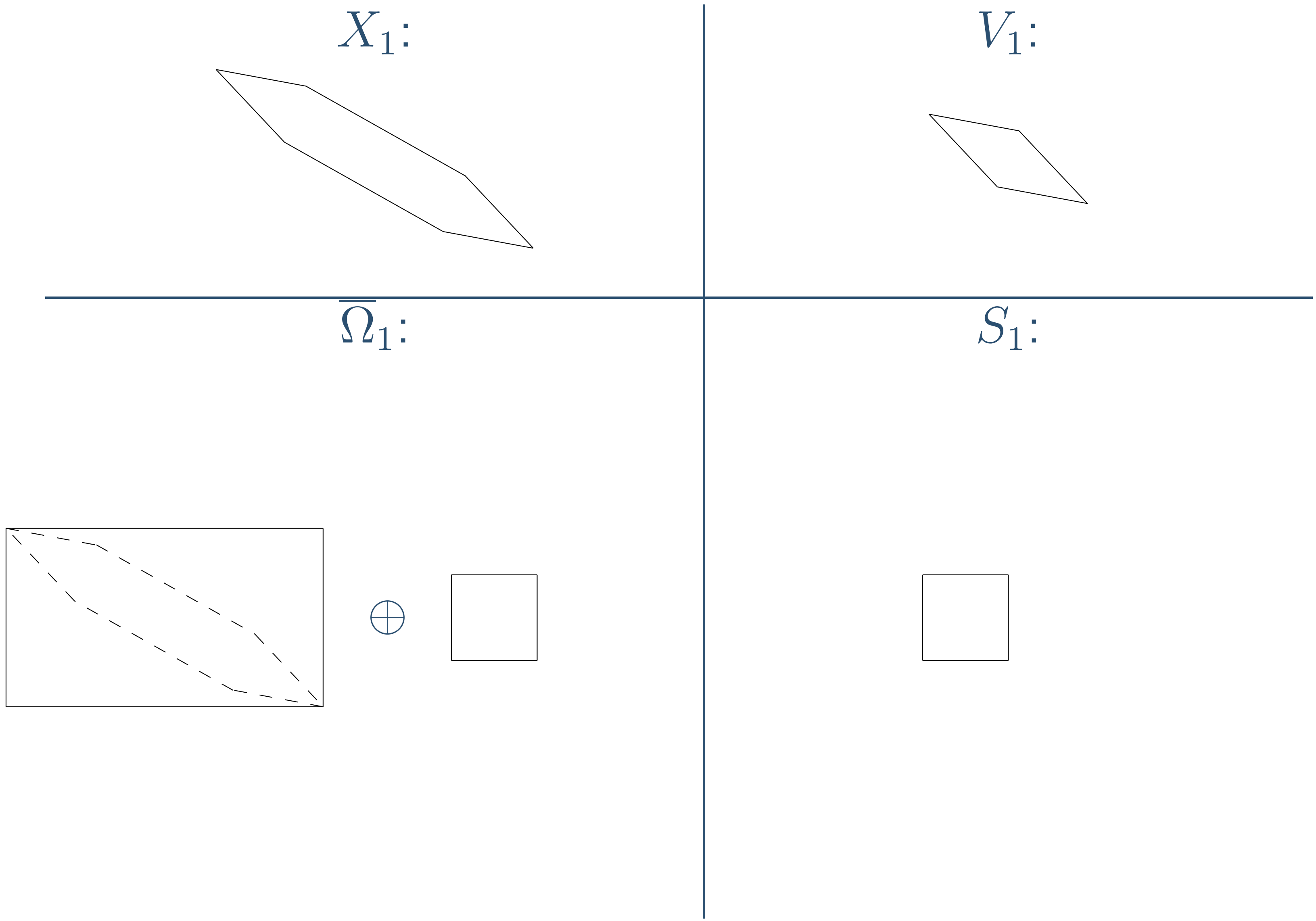
$\overline{\Omega}_1$:

$\mathrm{BOX}(X_1) \oplus S_1$

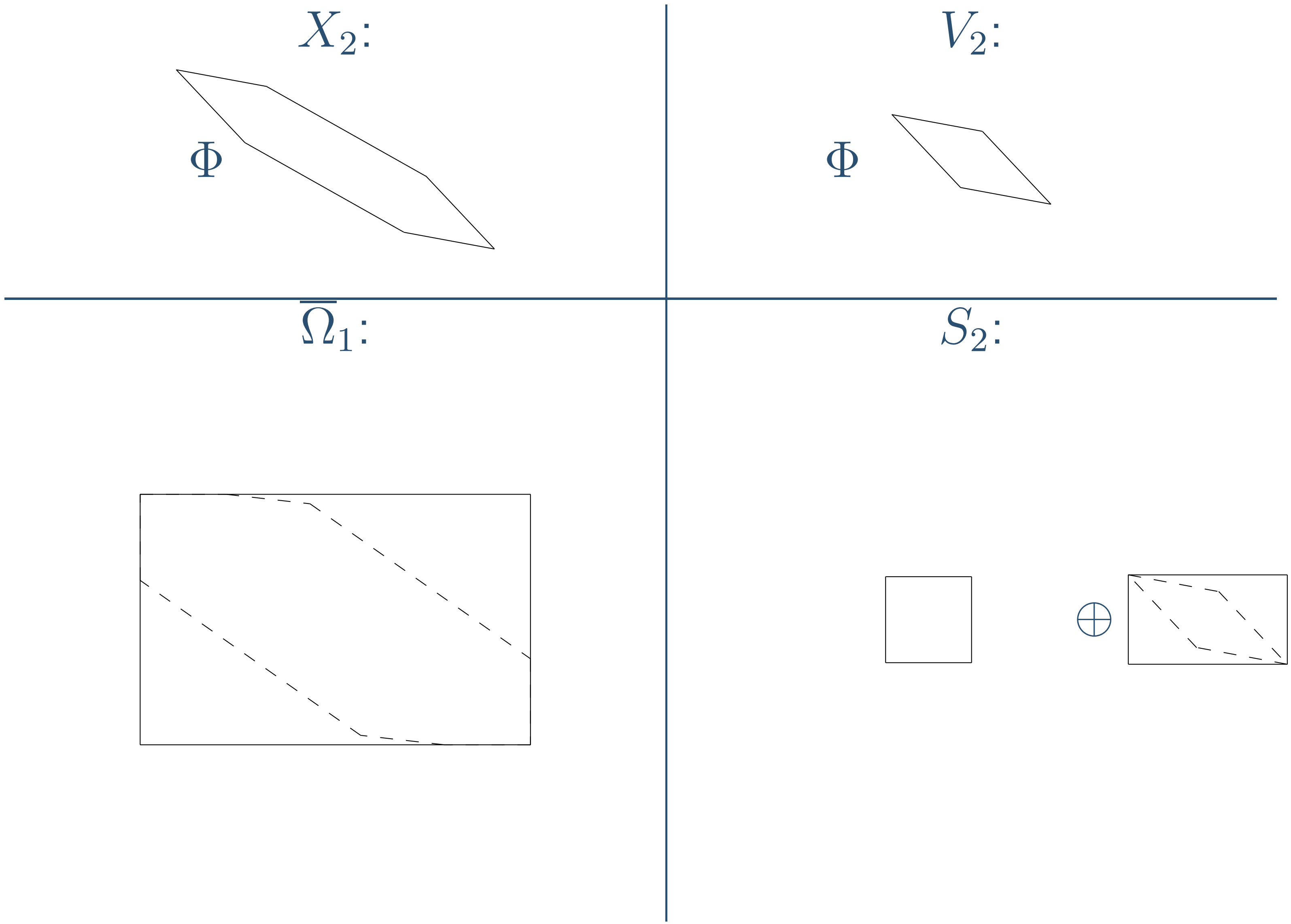
S_1:

$\bullet \quad \oplus$ 

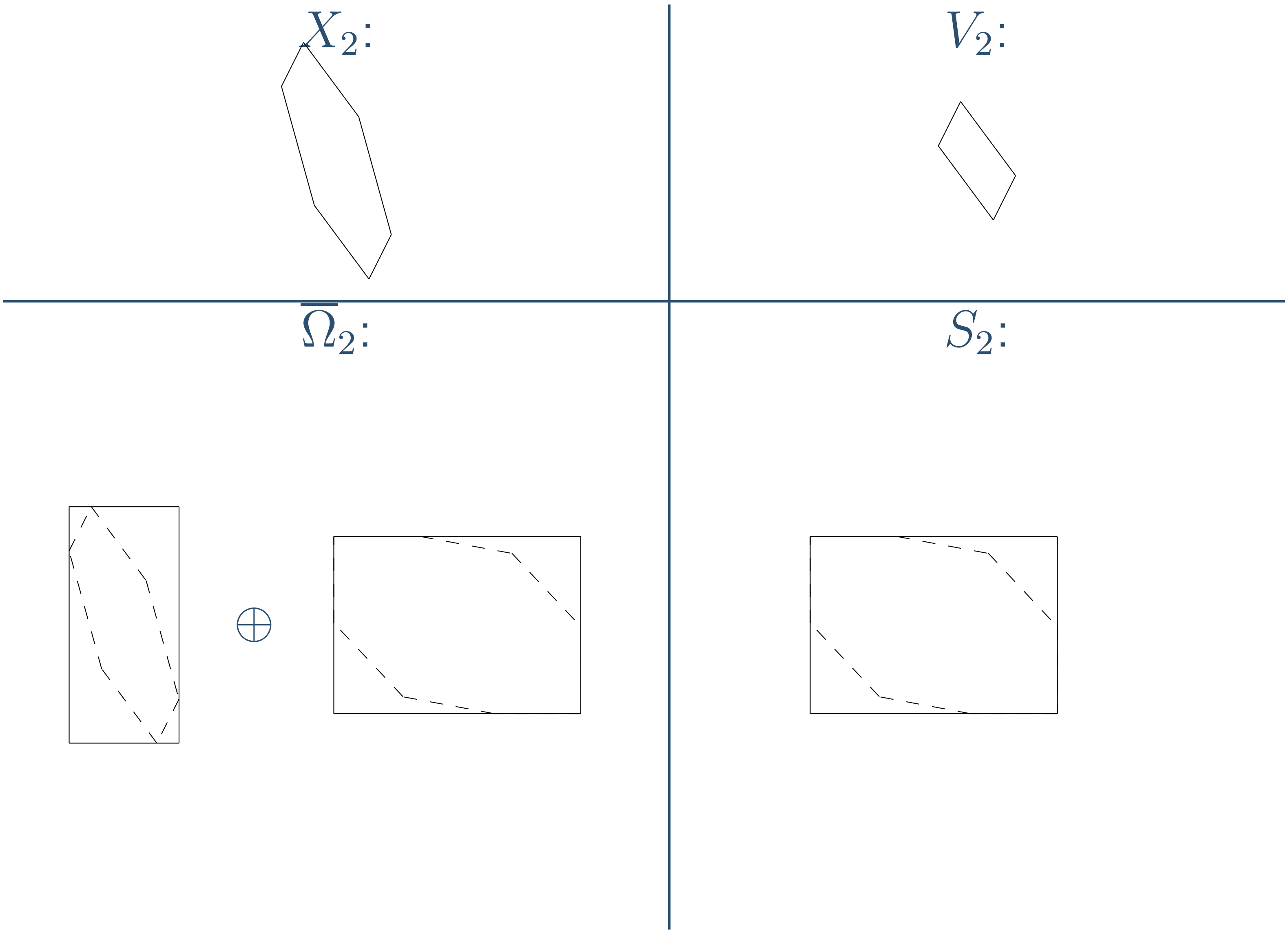
Example



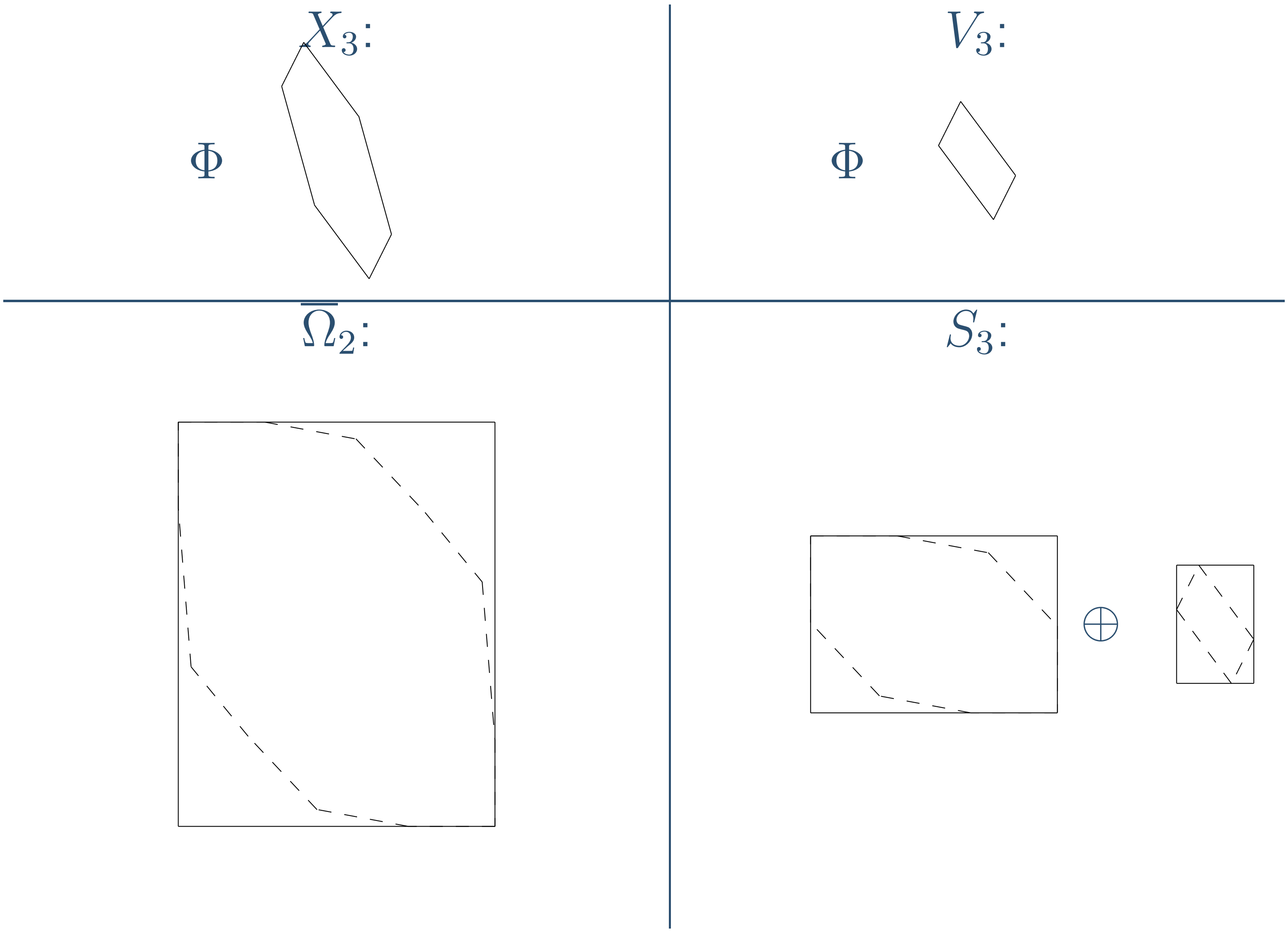
Example



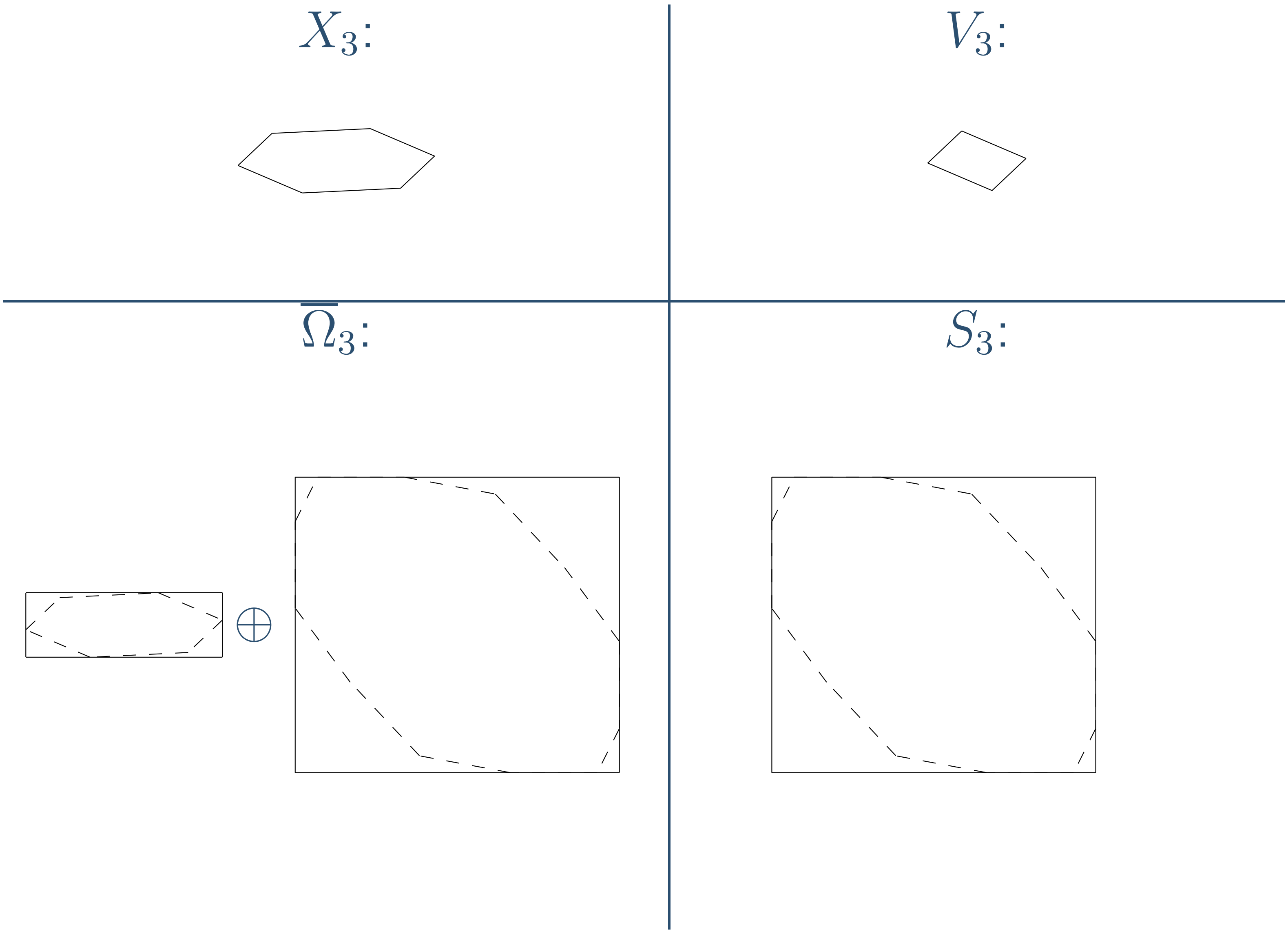
Example



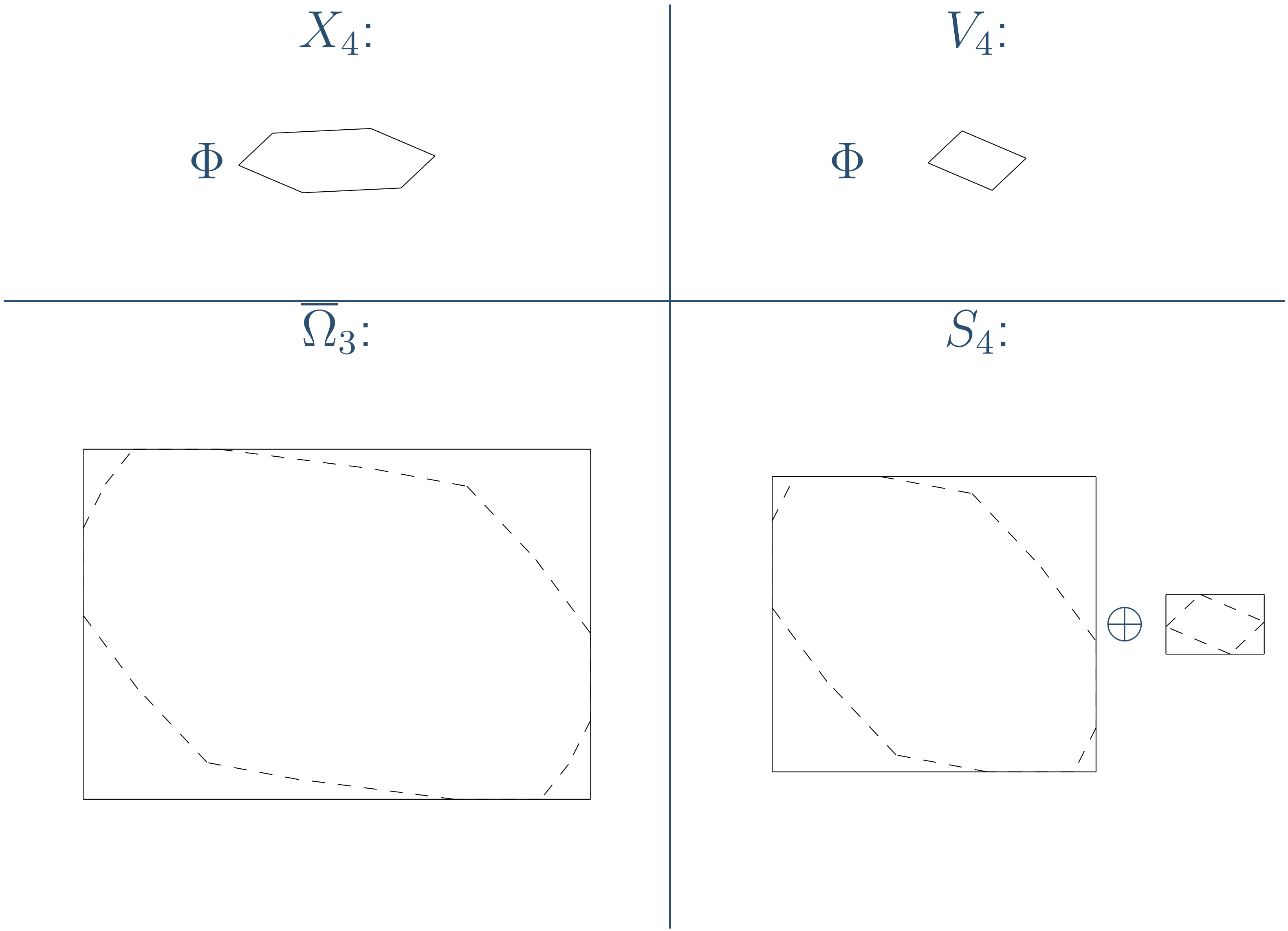
Example



Example

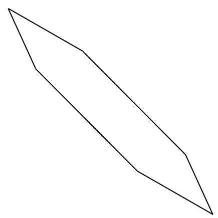


Example

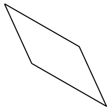


Example

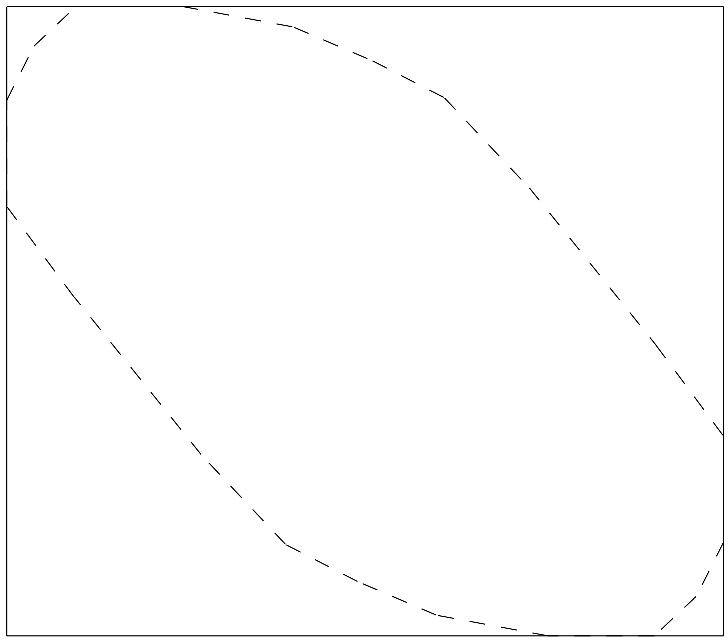
X_4:



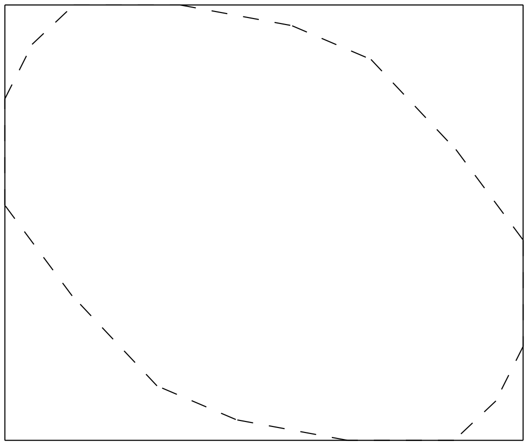
V_4:



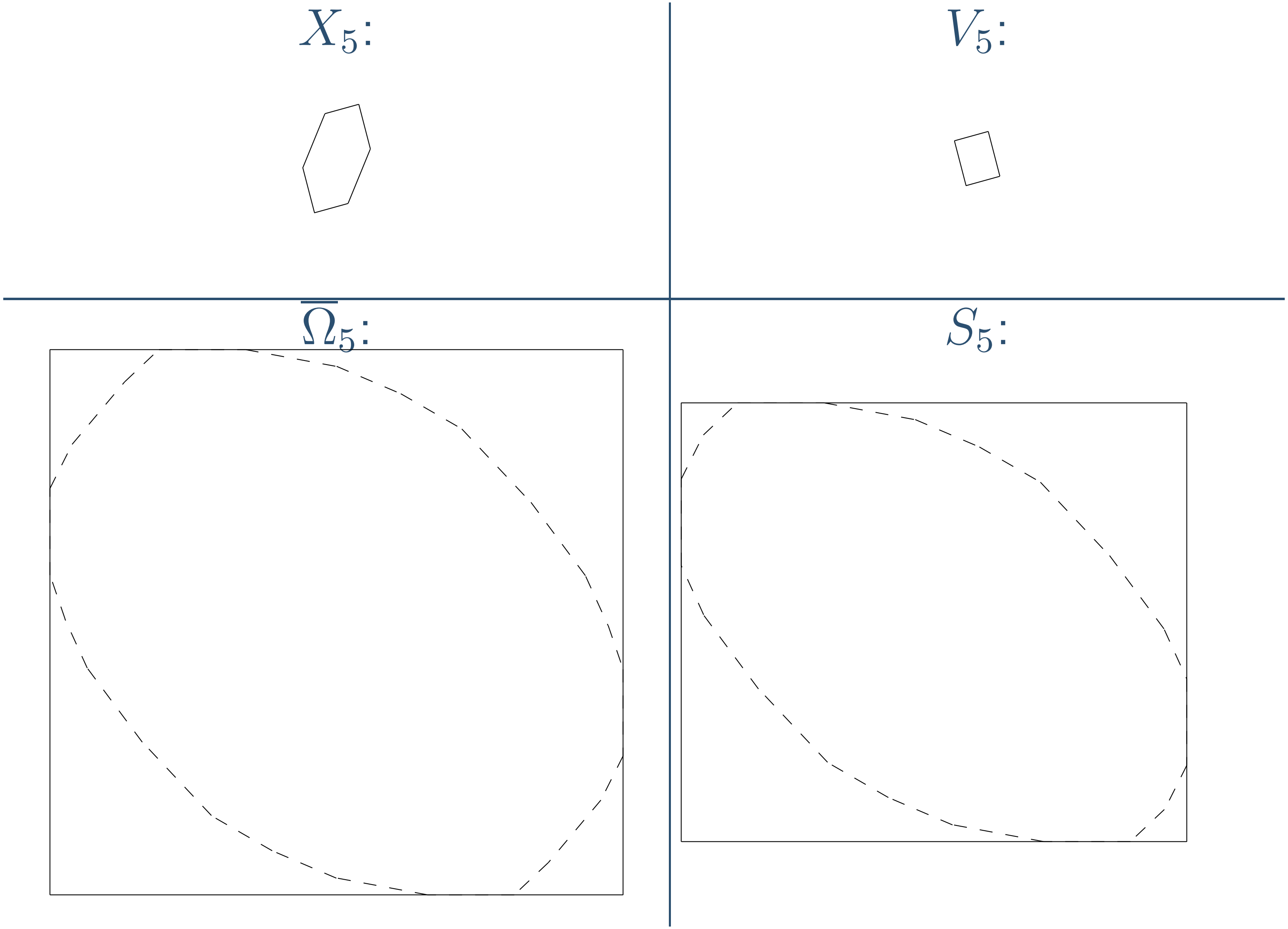
$\overline{\Omega}_4$:



S_4:



Example



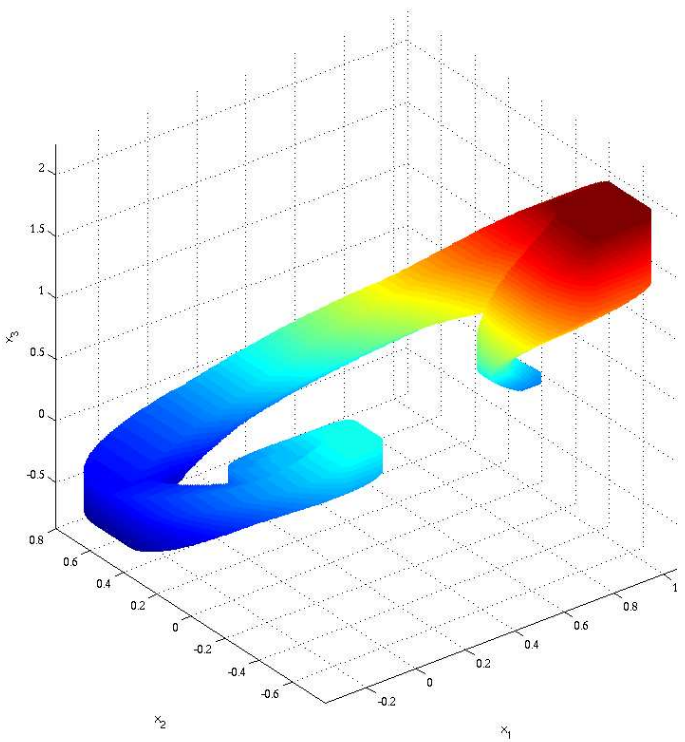
If we are tight in the direction given by the normal to the guards:

$$\overline{\Omega}_i \text{ intersects } G_e \Longleftrightarrow \Omega_i \text{ intersects } G_e.$$

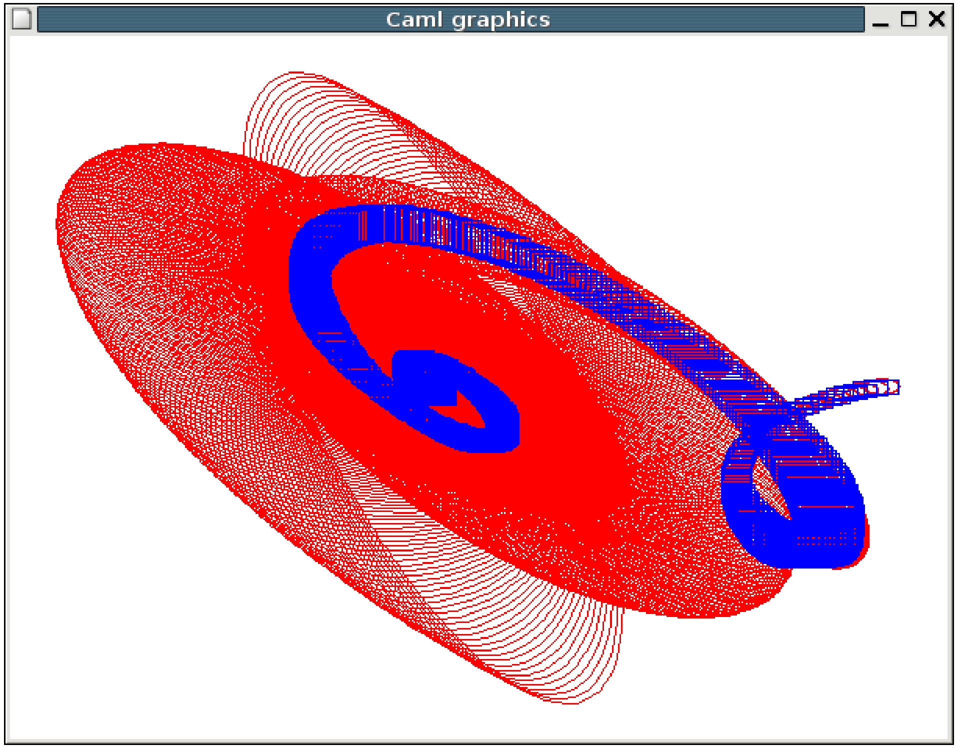
Experimental Results

Dim 5

Result can be exported to the Multi-Parametric Toolbox (MPT).



Interval Hull vs ET (tight in one random direction) [Kurzhanskiy,Varaiya]



dimension 5, 1000 time steps in 0.01s.

Benchmarks

$d =$	5	10	20	50	100	150	200
Exact	0.0s	0.02s	0.11s	1.11s	8.43s	35.9s	136s
BOX	0.0s	0.01s	0.07s	0.91s	8.08s	28.8s	131s

$d =$	5	10	20	50	100	150	200
Exact	246KB	492KB	1.72MB	8.85MB	33.7MB	75.2MB	133MB
BOX	246KB	246KB	246KB	492KB	983KB	2.21MB	3.69MB

Table 1: Time and memory consumption for $N = 100$ for several linear time-invariant systems of different dimensions

Conclusion

Summary

- as fast as Kurzhanskiy and Varaiya's algorithm (tight in two directions)
- needs very little memory
- can deal with any kind of input
- can produce nearly any kind of output (polytopes, ellipsoids,. . .)
- tight over- and under-approximation in user specified directions
  - better approximation
  - guard optimal

Future work

- implementation of $\mathcal{S}$-band intersections
- intersection with the guards
- use of the support function
  - drop complexity
  - parallelization