Numerical Abstract Domain using Support Function.

Yassamine Seladji
and
Olivier Bouissou.
CEA, LIST, LMeASI.
yassamine.seladji@cea.fr
olivier.bouissou@cea.fr
19 juin 2012

An industriel problem

- The crash of Ariane 5 : caused by an overflow.
  $\Longrightarrow$ 700 Million euro of lost.

Program

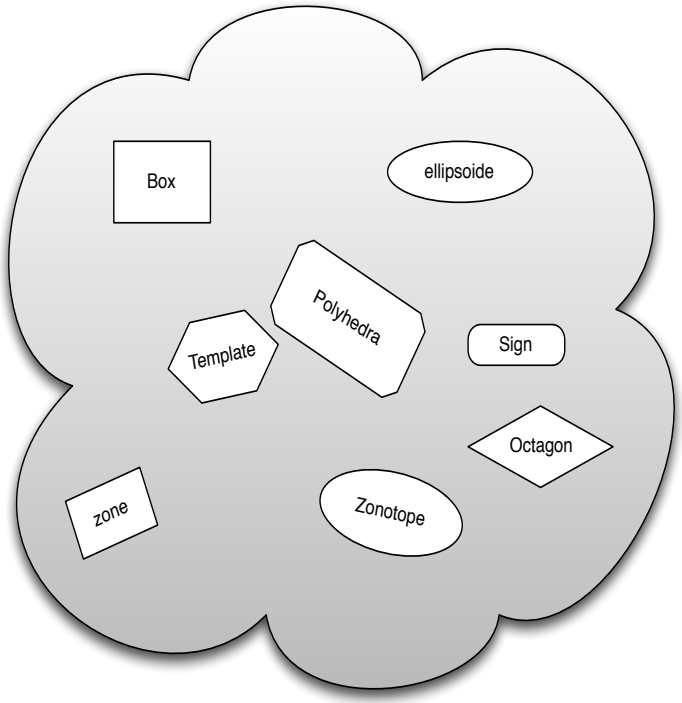
Input : $S_0 \subseteq \mathbb{R}^n$
Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.
Input : $c \in \mathbb{R}^n$, $l \in \mathbb{R}$
$X \in S_0$
while $(\langle X, c \rangle \leq l)$ {
$\quad X = AX + b.$
}

Program

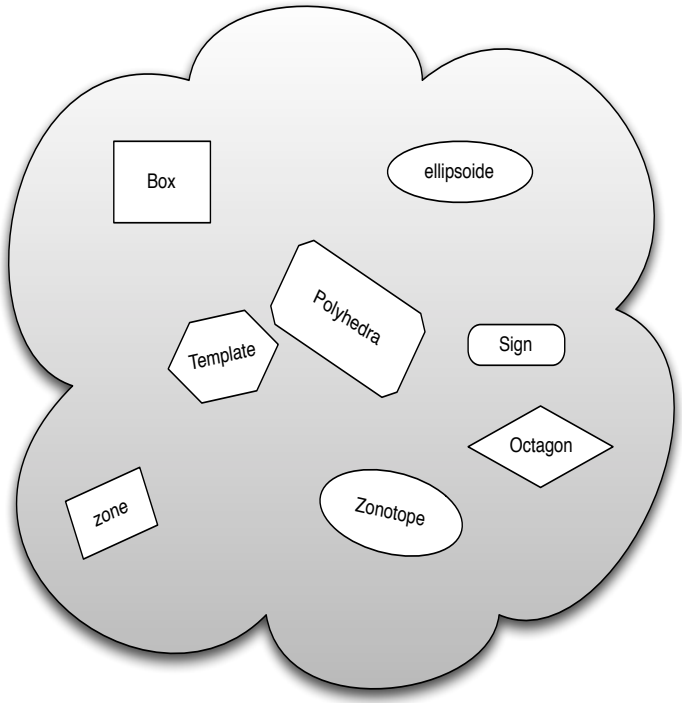
Input : $S_0 \subseteq \mathbb{R}^n$
Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.
Input : $c \in \mathbb{R}^n$, $l \in \mathbb{R}$
$X \in S_0$
while $(\langle X, c \rangle \leq l)$ {
$\quad X = AX + b.$
}

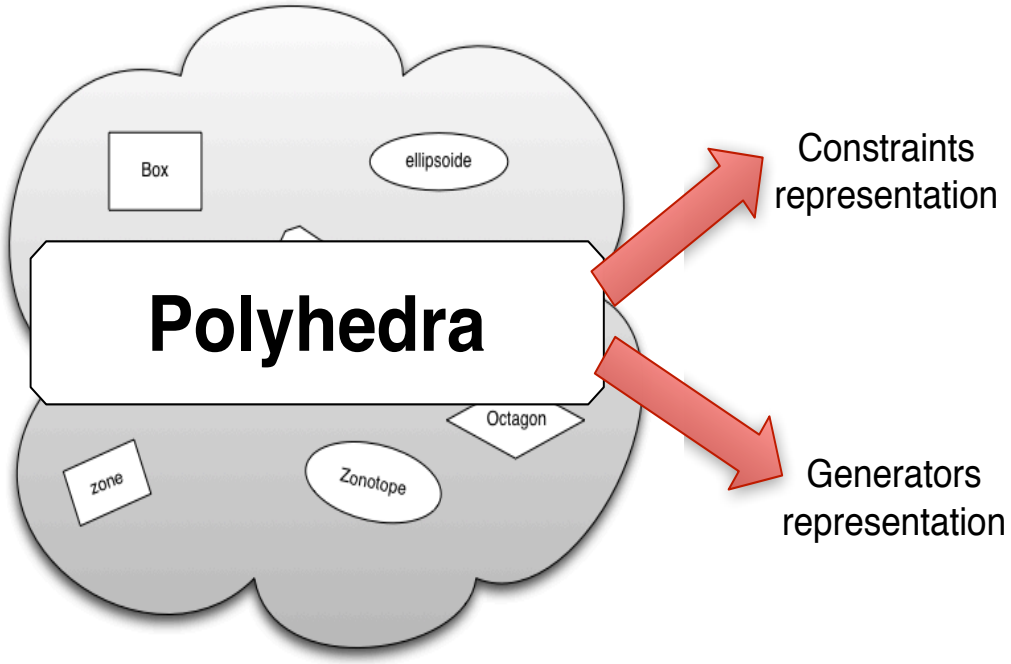
$$S_i = S_{i-1} \cup [(AS_{i-1} + b) \cap (\langle c, X \rangle \leq l)]$$

Static analysis by abstract interpretation

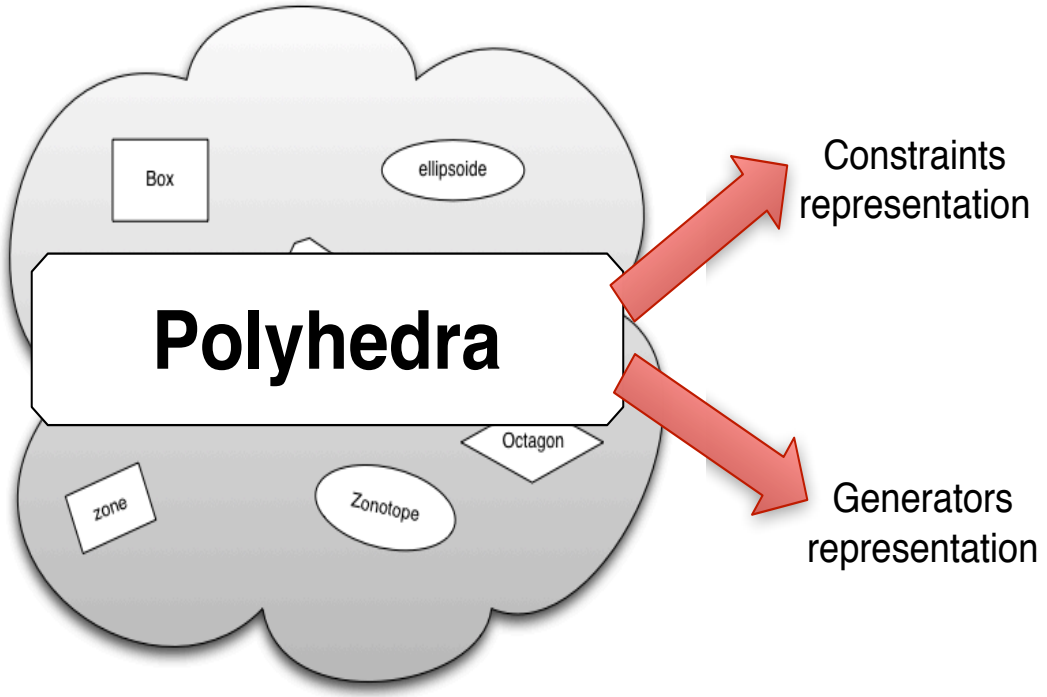


Static analysis by abstract interpretation

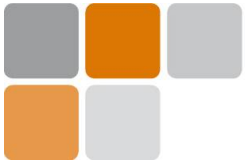



Box
ellipsoide
Polyhedra
Octagon
zone
Zonotope
Constraints representation
Generators representation

Static analysis by abstract interpretation



Support function

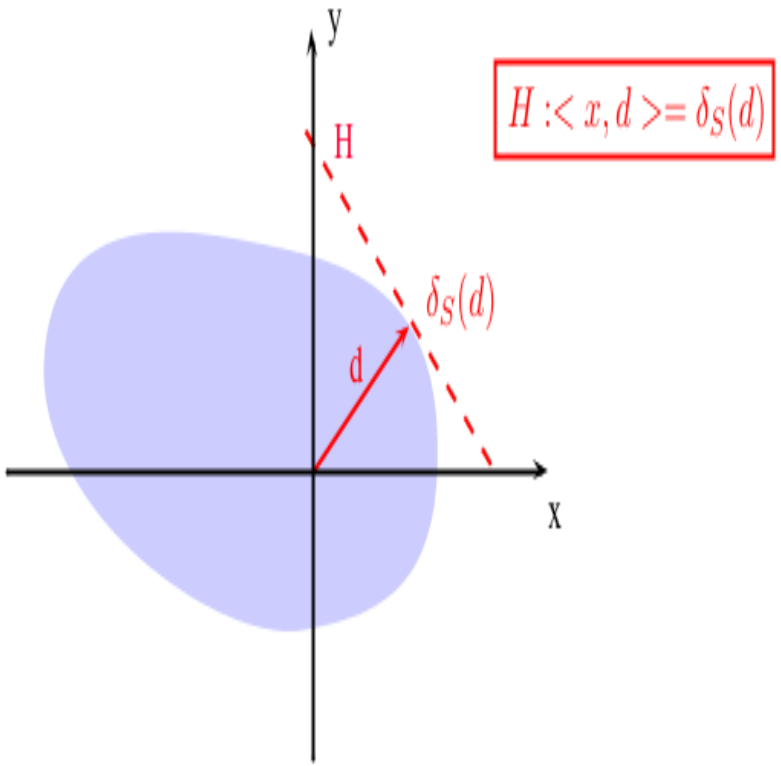


Outline

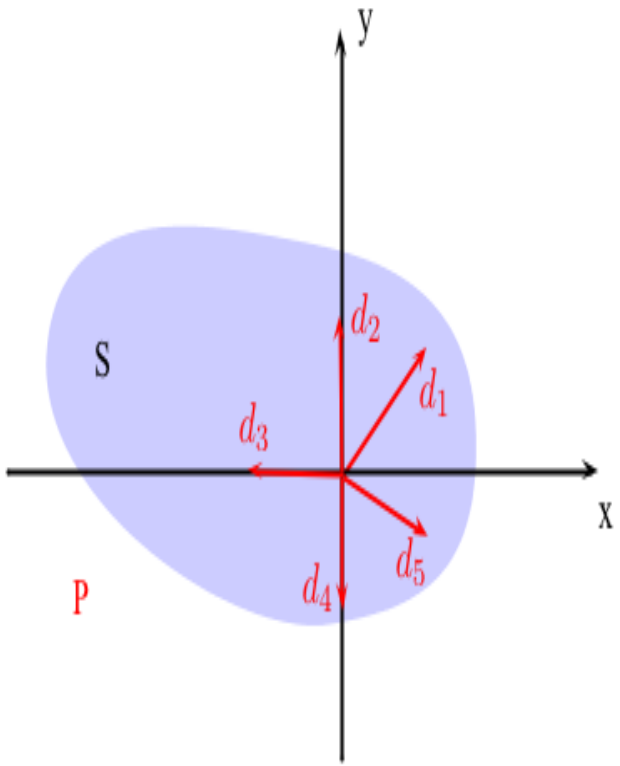
Definition

Let S be a convex set and δ_S its support function, such that :

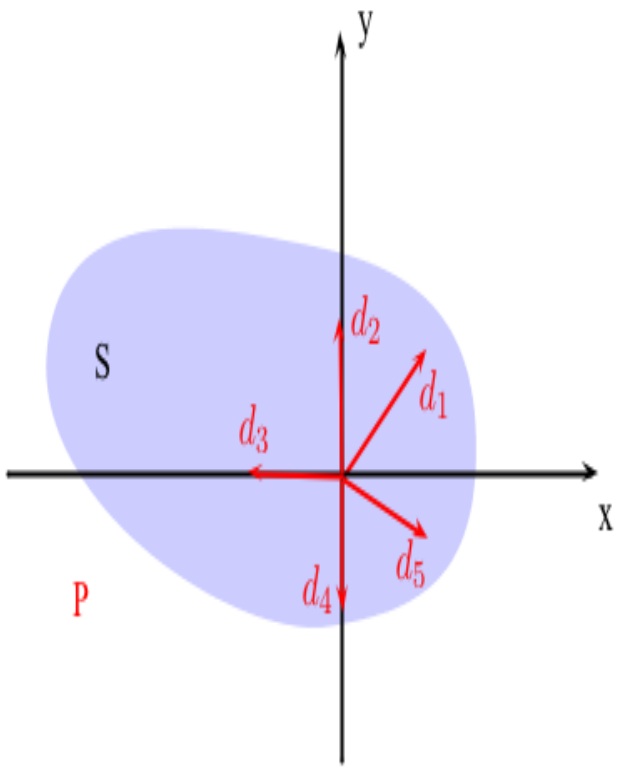
$$\forall d \in \mathbb{R}^n, \delta_S(d) = sup\{\langle x, d\rangle : x \in S\}$$



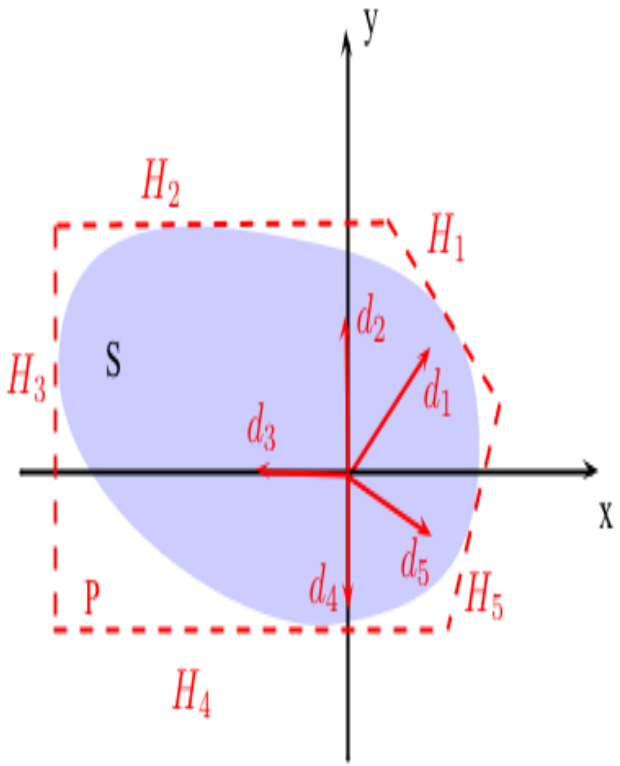
Let $\Delta = \{d_1, d_2, d_3, d_4, d_5\}$ be a set of directions.



Let $\Delta = \{d_1, d_2, d_3, d_4, d_5\}$ be a set of directions.



Let $\Delta = \{d_1, d_2, d_3, d_4, d_5\}$ be a set of directions.



Property

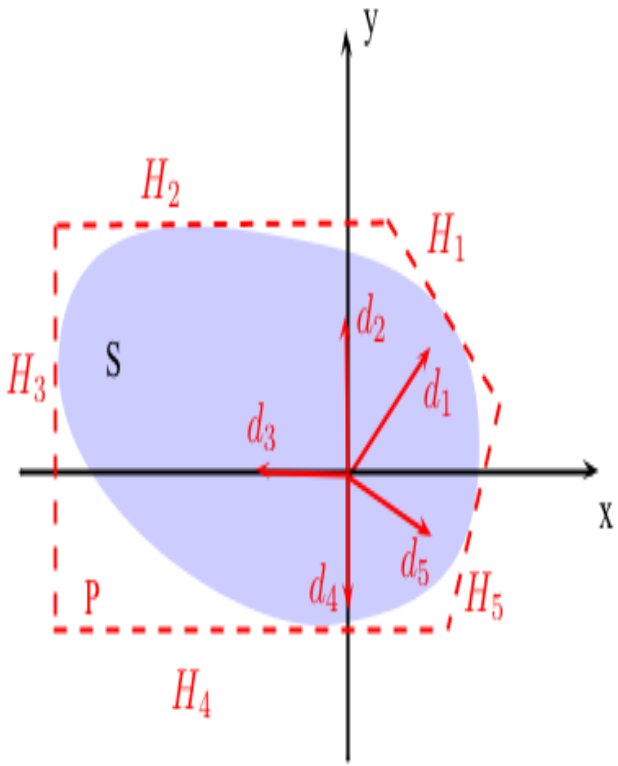
Let S be a convex set, and $\Delta \subseteq \mathbb{R}^n$ be a set of directions.
We put

$$P = \bigcap_{d \in \Delta} \{x \in \mathbb{R}^n | \langle x, d \rangle \leq \delta_S(d)\}$$

Then

$$S \subseteq P$$

Let $\Delta = \{d_1, d_2, d_3, d_4, d_5\}$ be a set of directions.



The special case of polyhedron

Let $\mathbb{P}$ be a polyhedron. If $\mathbb{P}$ is represented by :

- Linear system, $\delta_{\mathbb{P}}$ is obtained using Linear Programming.
- Generators (vertices) v_i, $\delta_{\mathbb{P}}(d) = sup\{\langle v_i, d\rangle : v_i \in \mathbb{P}\}$.

Properties

Let S, S' be two convex sets. We have :

- $\forall M \in \mathbb{R}^n \times \mathbb{R}^m, \delta_{MS}(d) = \delta_S(M^T d)$.
- $\forall \lambda \geq 0, \delta_{\lambda S}(d) = \lambda \delta_S(d)$.
- $\delta_{S \oplus S'}(d) = \delta_S(d) + \delta_{S'}(d)$.
- $\delta_{S \cup S'}(d) = \max(\delta_S(d), \delta_{S'}(d))$.
- $\delta_{S \cap S'}(d) \leq \min(\delta_S(d), \delta_{S'}(d))$.

Properties

Let S, S' be two convex sets. We have :

- $\forall M \in \mathbb{R}^n \times \mathbb{R}^m, \delta_{MS}(d) = \delta_S(M^T d)$.
- $\forall \lambda \geq 0, \delta_{\lambda S}(d) = \lambda \delta_S(d)$.
- $\delta_{S \oplus S'}(d) = \delta_S(d) + \delta_{S'}(d)$.$S \oplus S' = \{x + x' \mid x \in S,\ x' \in S'\}$
- $\delta_{S \cup S'}(d) = \max(\delta_S(d), \delta_{S'}(d))$.
- $\delta_{S \cap S'}(d) \leq \min(\delta_S(d), \delta_{S'}(d))$.

Properties

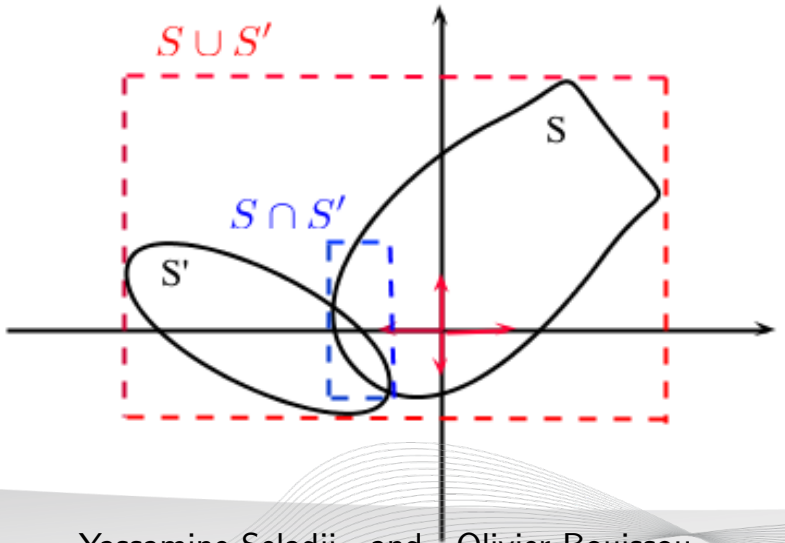
Let S, S' be two convex sets. We have :

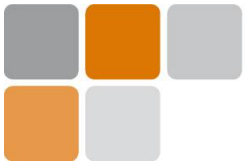
- $\forall M \in \mathbb{R}^n \times \mathbb{R}^m, \delta_{MS}(d) = \delta_S(M^T d)$.
- $\forall \lambda \geq 0, \delta_{\lambda S}(d) = \lambda \delta_S(d)$.
- $\delta_{S \oplus S'}(d) = \delta_S(d) + \delta_{S'}(d)$.
- $\delta_{S \cup S'}(d) = \max(\delta_S(d), \delta_{S'}(d))$.
- $\delta_{S \cap S'}(d) \leq \min(\delta_S(d), \delta_{S'}(d))$.

Properties

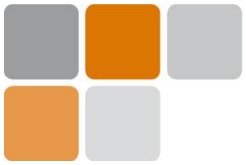
Let S, S' be two convex sets. We have :

- $\forall M \in \mathbb{R}^n \times \mathbb{R}^m, \delta_{MS}(d) = \delta_S(M^T d)$.
- $\forall \lambda \geq 0, \delta_{\lambda S}(d) = \lambda \delta_S(d)$.
- $\delta_{S \oplus S'}(d) = \delta_S(d) + \delta_{S'}(d)$.
- $\delta_{S \cup S'}(d) = \max(\delta_S(d), \delta_{S'}(d))$.
- $\delta_{S \cap S'}(d) \leq \min(\delta_S(d), \delta_{S'}(d))$.





For a set of directions Δ,



For a set of directions Δ, let $\mathbb{P}^{\sharp}_{\Delta} = \Delta \to \mathbb{R}_{\infty}$ be the abstract domain.

For a set of directions Δ, let $\mathbb{P}^{\sharp}_{\Delta} = \Delta \to \mathbb{R}_{\infty}$ be the abstract domain.

The concretisation function

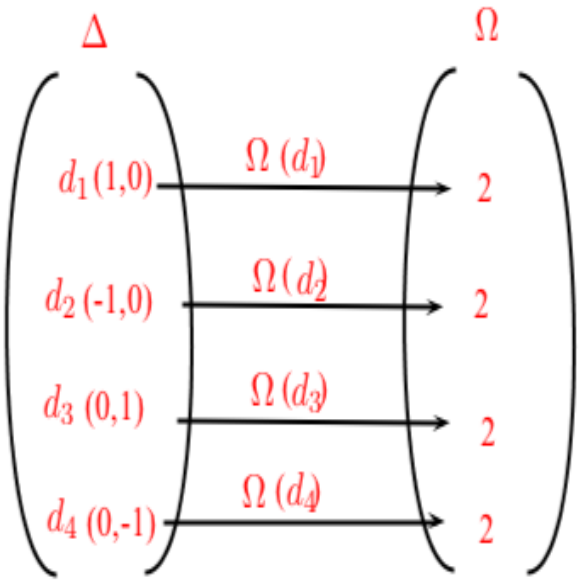
$$\begin{array}{rccl} \gamma_{\Delta} : & (\Delta \to \mathbb{R}_{\infty}) & \longrightarrow & P(\mathbb{R}^n) \\ & \Omega & \longrightarrow & \bigcap_{d \in \Delta} \{x \in \mathbb{R}^n \mid \langle x, d \rangle \leq \Omega(d)\} \end{array}$$

For a set of directions Δ, let $\mathbb{P}^{\sharp}_{\Delta} = \Delta \to \mathbb{R}_{\infty}$ be the abstract domain.

The concretisation function

$$\begin{array}{rccl} \gamma_{\Delta} : & (\Delta \to \mathbb{R}_{\infty}) & \longrightarrow & P(\mathbb{R}^n) \\ & \Omega & \longrightarrow & \bigcap_{d \in \Delta} \{x \in \mathbb{R}^n \mid \langle x, d \rangle \leq \Omega(d)\} \end{array}$$

Example :

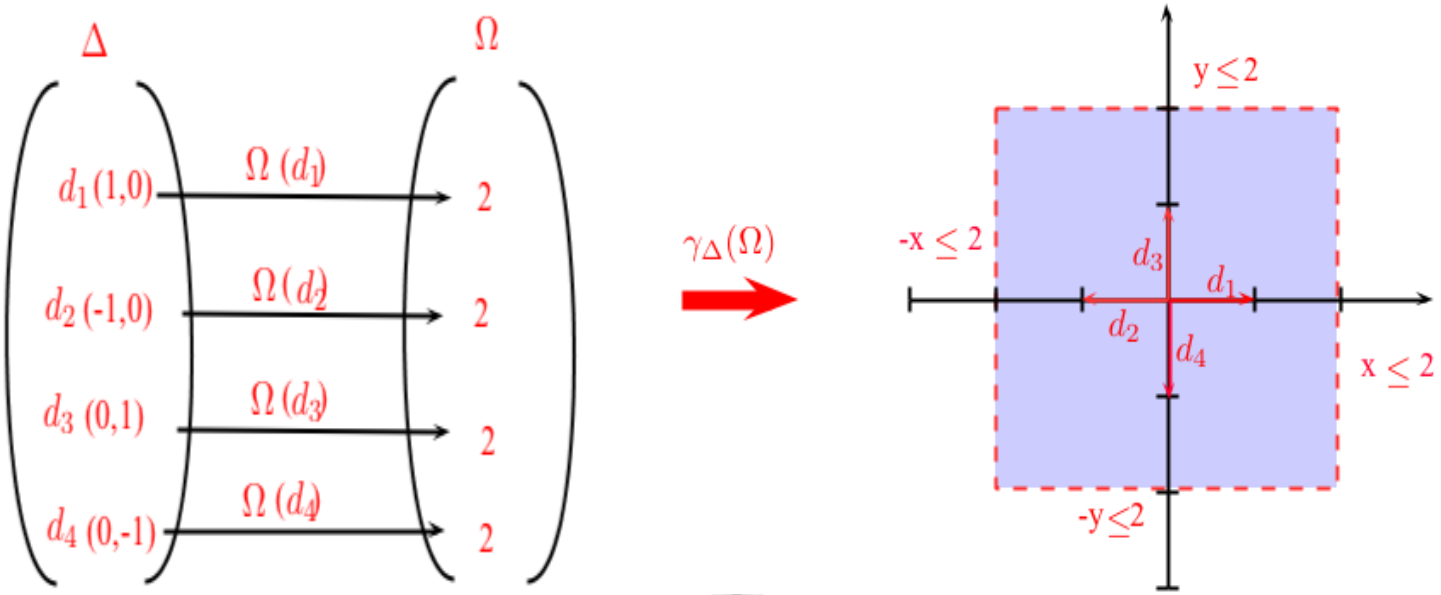


For a set of directions Δ, let $\mathbb{P}^{\sharp}_{\Delta} = \Delta \to \mathbb{R}_{\infty}$ be the abstract domain.

The concretisation function

$$\begin{array}{rccl} \gamma_{\Delta} : & (\Delta \to \mathbb{R}_{\infty}) & \longrightarrow & P(\mathbb{R}^n) \\ & \Omega & \longrightarrow & \bigcap_{d \in \Delta} \{x \in \mathbb{R}^n \mid \langle x, d \rangle \leq \Omega(d)\} \end{array}$$

Example :



The abstraction function

$$\alpha_\Delta : \quad P(\mathbb{R}^n) \longrightarrow (\Delta \to \mathbb{R}_\infty)$$

$$S \longrightarrow \begin{cases} \lambda d.\ -\infty & \text{if } S = \emptyset \\ \lambda d.\ +\infty & \text{if } S = \mathbb{R}^n \\ \lambda d.\ \delta_S(d) & \text{otherwise} \end{cases}$$

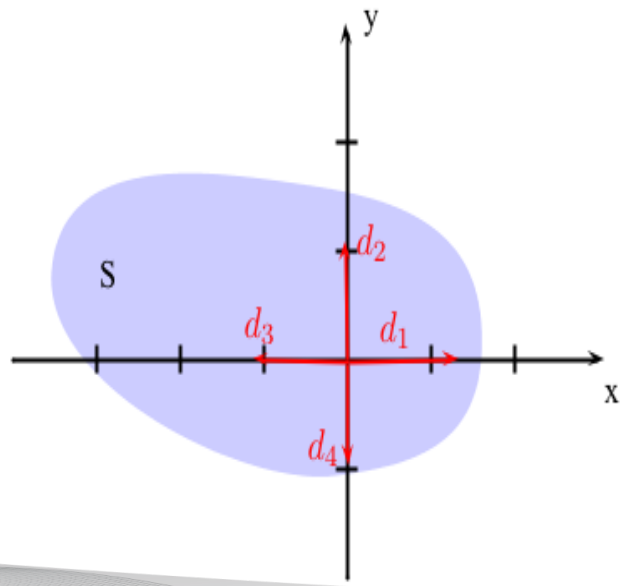
Example :

The abstraction function

$$\alpha_\Delta : \quad P(\mathbb{R}^n) \longrightarrow (\Delta \to \mathbb{R}_\infty)$$

$$S \longrightarrow \begin{cases} \lambda d.\ -\infty & \text{if } S = \emptyset \\ \lambda d.\ +\infty & \text{if } S = \mathbb{R}^n \\ \lambda d.\ \delta_S(d) & \text{otherwise} \end{cases}$$

Example :

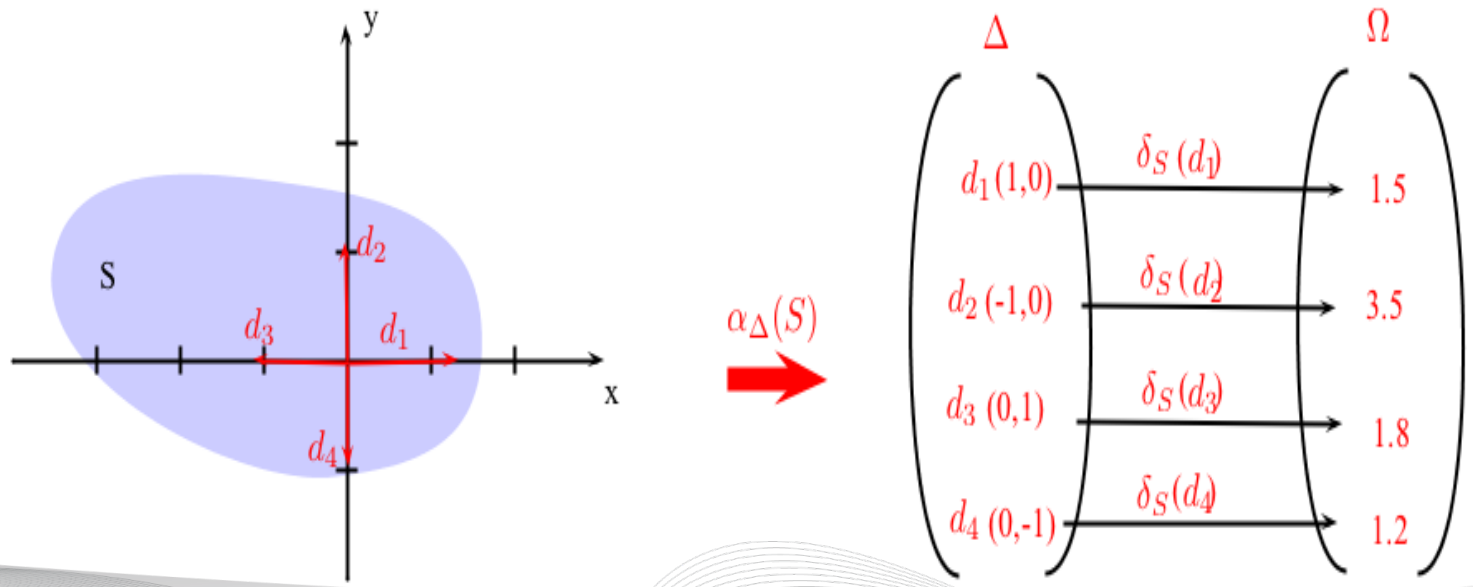


The abstraction function

$$\alpha_\Delta : \quad P(\mathbb{R}^n) \longrightarrow (\Delta \to \mathbb{R}_\infty)$$

$$S \longrightarrow \begin{cases} \lambda d.\ -\infty & \text{if } S = \emptyset \\ \lambda d.\ +\infty & \text{if } S = \mathbb{R}^n \\ \lambda d.\ \delta_S(d) & \text{otherwise} \end{cases}$$

Example :

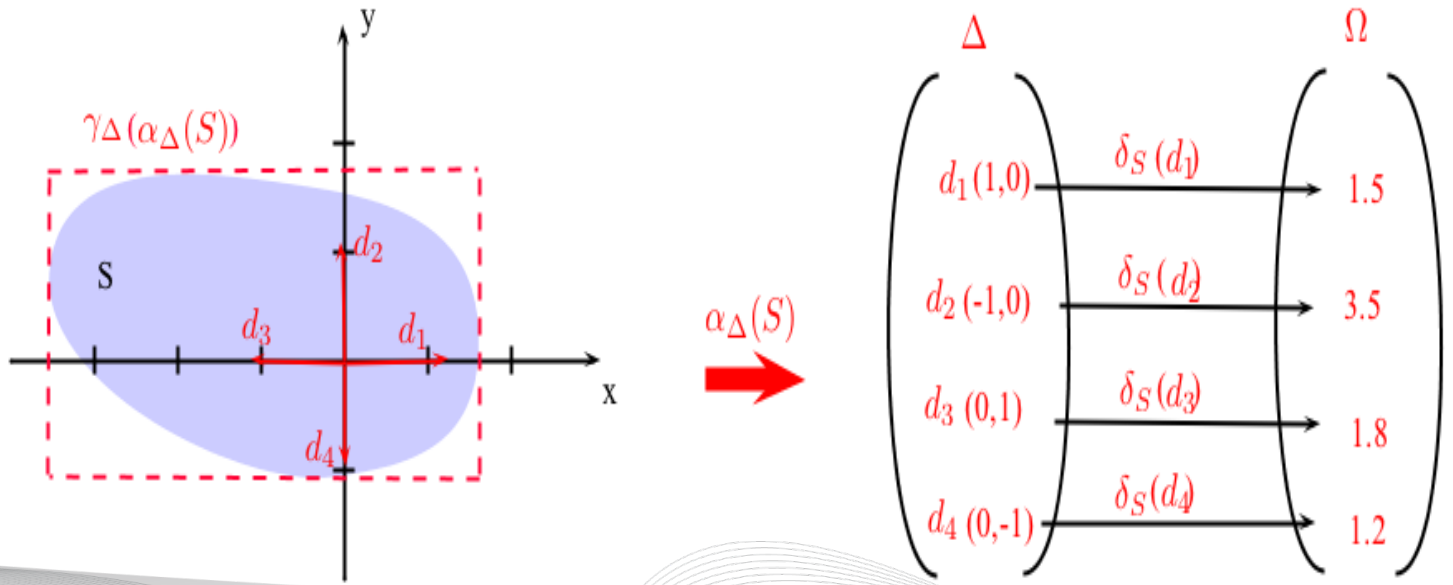


The abstraction function

$$\alpha_\Delta : \quad P(\mathbb{R}^n) \longrightarrow (\Delta \to \mathbb{R}_\infty)$$

$$S \longrightarrow \begin{cases} \lambda d.\ -\infty & \text{if } S = \emptyset \\ \lambda d.\ +\infty & \text{if } S = \mathbb{R}^n \\ \lambda d.\ \delta_S(d) & \text{otherwise} \end{cases}$$

Example :



The complete lattice $\langle \mathbb{P}^{\sharp}_{\Delta}, \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$ is defined by :

- An order relation : $\Omega_1 \sqsubseteq \Omega_2 \Leftrightarrow \gamma_{\Delta}(\Omega_1) \subseteq \gamma_{\Delta}(\Omega_2)$.
- A minimal element : $\bot = \lambda d.\ -\infty$.
- A maximal element : $\top = \lambda d.\ +\infty$.
- A join operator : $\Omega_1 \sqcup \Omega_2 = \lambda d.\ \max(\Omega_1(d), \Omega_2(d))$.
- A meet operator : $\Omega_1 \sqcap \Omega_2 = \lambda d.\ \min(\Omega_1(d), \Omega_2(d))$.

The complete lattice $\langle \mathbb{P}^{\sharp}_{\Delta}, \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$ is defined by :

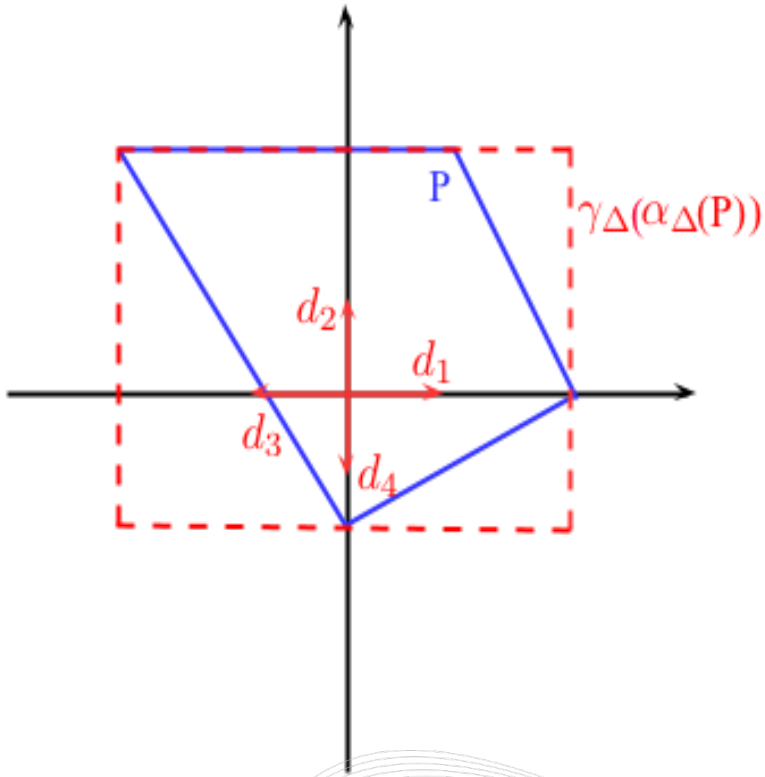
- An order relation : $\Omega_1 \sqsubseteq \Omega_2 \Leftrightarrow \gamma_{\Delta}(\Omega_1) \subseteq \gamma_{\Delta}(\Omega_2)$.
- A minimal element : $\bot = \lambda d.\ -\infty$.
- A maximal element : $\top = \lambda d.\ +\infty$.
- A join operator : $\Omega_1 \sqcup \Omega_2 = \lambda d.\ \max(\Omega_1(d), \Omega_2(d))$.
- A meet operator : $\Omega_1 \sqcap \Omega_2 = \lambda d.\ \min(\Omega_1(d), \Omega_2(d))$.

Notes :

$$\gamma_{\Delta}(\Omega_1 \sqcup \Omega_2) = \gamma_{\Delta}(\Omega_1) \sqcup \gamma_{\Delta}(\Omega_2).$$
$$\gamma_{\Delta}(\Omega_1 \sqcap \Omega_2) \sqsupseteq \gamma_{\Delta}(\Omega_1) \sqcap \gamma_{\Delta}(\Omega_2).$$

Property

Let P be a polyhedron and $\Omega \in \mathbb{P}^{\sharp}_{\Delta}$ such that $\Omega = \alpha_{\Delta}(P)$. We have that, $P \subseteq \gamma_{\Delta}(\Omega)$ where this over approximation is tight as the vertices of P touch the faces of $\gamma_{\Delta}(\Omega)$.



Program

Input : $\mathbb{P}_0$ a bounded polyhedron.

Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.

Input : $c \in \mathbb{R}^n$, $l \in \mathbb{R}$

$X \in \mathbb{P}_0$

while $(\langle X, c \rangle \leq l)$ {

$X = AX + b.$

}

Program

Input : $\mathbb{P}_0$ a bounded polyhedron.
Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.
Input : $c \in \mathbb{R}^n$, $l \in \mathbb{R}$
$X \in \mathbb{P}_0$
while $(\langle X, c \rangle \leq l)$ {
$X = AX + b.$
}

$$\Omega_i = \Omega_{i-1} \cup [(A\Omega_{i-1} + b) \cap (\langle c, X \rangle \leq l)]$$

- **Case 1 :** $\Omega = A\Omega_0 + b$

Program

Input : $\mathbb{P}_0$ a bounded polyhedron.
Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.
$X = AX + b$.

- **Case 1 :** $\Omega = A\Omega_0 + b$

Program

Input : $\mathbb{P}_0$ a bounded polyhedron.
Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.
$X = AX + b$.

The abstract element

$$\Omega \quad = \quad \lambda d.\delta_{A\mathbb{P}_0 \oplus b}(d)$$

- **Case 1 :** $\Omega = A\Omega_0 + b$

Program

Input : $\mathbb{P}_0$ a bounded polyhedron.
Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.
$X = AX + b$.

The abstract element

$$\begin{aligned} \Omega &= \lambda d.\delta_{A\mathbb{P}_0 \oplus b}(d) \\ &= \lambda d.\delta_{\mathbb{P}_0}(A^T d) + \langle b, d \rangle \end{aligned}$$

- **Case 2 :** $\Omega_i = \Omega_{i-1} \cup [(A\Omega_{i-1} + b) \cap (\langle c, X \rangle \not\# I)]$

Program

Input : $\mathbb{P}_0$ a bounded polyhedron.
Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.
while $(true)$ {
$X = AX + b$
}

- **Case 2 :** $\Omega_i = \Omega_{i-1} \cup [(A\Omega_{i-1} + b) \cap (\langle c, X \rangle \not\# I)]$

Program

Input : $\mathbb{P}_0$ a bounded polyhedron.
Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.
while $(true)$ {
$X = AX + b$
}

The abstract element

$\Omega_i = \lambda d. \max\{\delta_{\mathbb{P}_0}(A^{Tj}d) + \sum_{k=1}^{j} \langle b, A^{T(k-1)}d \rangle, j = 0, .., i\}$

- **Case 2 :** $\Omega_i = \Omega_{i-1} \cup [(A\Omega_{i-1} + b) \cap (\langle c, X \rangle \not\bowtie I)]$

Program

Input : $\mathbb{P}_0$ a bounded polyhedron.
Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.
while $(true)$ {
$X = AX + b$
}

$\alpha_\Delta(\mathbb{P}_i) = \Omega_i$

The abstract element

$\Omega_i = \lambda d. \max\{\delta_{\mathbb{P}_0}(A^{Tj}d) + \sum_{k=1}^{j} \langle b, A^{T(k-1)}d \rangle, j = 0, .., i\}$

- **Case 3 :** $\Omega_i = \Omega_{i-1} \cup [(A\Omega_{i-1} + b) \cap (\langle c, X \rangle \leq l)]$

Program

Input : $\mathbb{P}_0$ a bounded polyhedron.
Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.
Input : $c \in \mathbb{R}^n$, $l \in \mathbb{R}$
while $(\langle X, c \rangle \leq l)$ {
$X = AX + b.$
}

- **Case 3 :** $\Omega_i = \Omega_{i-1} \cup [(A\Omega_{i-1} + b) \cap (\langle c, X \rangle \leq l)]$

Program

Input : $\mathbb{P}_0$ a bounded polyhedron.
Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.
Input : $c \in \mathbb{R}^n$, $l \in \mathbb{R}$
while $(\langle X, c \rangle \leq l)$ {
$X = AX + b.$
}

$H : \langle X, c \rangle \leq l$

- **Case 3 :** $\Omega_i = \Omega_{i-1} \cup [(A\Omega_{i-1} + b) \cap (\langle c, X \rangle \leq l)]$

Program

Input : $\mathbb{P}_0$ a bounded polyhedron.
Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.
Input : $c \in \mathbb{R}^n$, $l \in \mathbb{R}$
while $(\langle X, c \rangle \leq l)$ {
$X = AX + b.$
}

$$H : \langle X, c \rangle \leq l$$

$$\delta_H(d) = \begin{cases} l & \text{if } d = \lambda c \\ +\infty & \text{otherwise} \end{cases}$$

- **Case 3 :** $\Omega_i = \Omega_{i-1} \cup [(A\Omega_{i-1} + b) \cap (\langle c, X \rangle \leq l)]$

Program

Input : $\mathbb{P}_0$ a bounded polyhedron.
Input : $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$.
Input : $c \in \mathbb{R}^n$, $l \in \mathbb{R}$
while $(\langle X, c \rangle \leq l)$ {
$X = AX + b.$
}

$$H : \langle X, c \rangle \leq l$$

$$\delta_H(d) = \begin{cases} l & \text{if } d = \lambda c \\ +\infty & \text{otherwise} \end{cases}$$

We put $\Delta \cup \{c\}$:

- $\Delta_1 = \{c\} \cup \{d \in \Delta | d = \lambda c, \lambda \geq 0\}$.
- $\Delta_2 = \Delta \setminus \Delta_1$

- **Case 3 :** $\Omega_i = \Omega_{i-1} \cup [(A\Omega_{i-1} + b) \cap (\langle c, X \rangle \leq l)]$
- For $d \in \Delta_2$: we use the same method as for **Case 2**.

- **Case 3 :** $\Omega_i = \Omega_{i-1} \cup [(A\Omega_{i-1} + b) \cap (\langle c, X \rangle \leq l)]$
- For $d \in \Delta_2$: we use the same method as for **Case 2**.

$$\Omega_i = \lambda d. \max\{\delta_{\mathbb{P}_0}(A^{Tj}d) + \sum_{k=1}^{j} \langle b, A^{T(k-1)}d \rangle, j = 0, .., i\}$$

- **Case 3 :** $\Omega_i = \Omega_{i-1} \cup [(A\Omega_{i-1} + b) \cap (\langle c, X \rangle \leq l)]$
- For $d \in \Delta_2$: we use the same method as for **Case 2**.

$$\Omega_i = \lambda d. \max\{\delta_{\mathbb{P}_0}(A^{Tj} d) + \sum_{k=1}^{j} \langle b, A^{T(k-1)} d \rangle, j = 0, .., i\}$$

- For $d \in \Delta_1$, we have that :

$$\Omega_i(d) = \max(\delta_{\gamma_\Delta(\Omega_{i-1})}(d), \min(\delta_{\gamma_\Delta(\Omega_{i-1})}(A^T d) + \langle b, d \rangle, l))$$

Such that $\lambda d. \delta_{\mathbb{P}_i}(d) \leq \Omega_i(d)$.

$$\Omega_i(d) = \max\{\delta_{\mathbb{P}_0}(A^{Tj}d) + \sum_{k=1}^{j}\langle b, A^{T(k-1)}d\rangle, j = 0, .., i\}$$

Algorithm 1 Kleene Algorithm using support function.

Require: $\Delta \subset \mathbb{R}^n$, set of l directions. $\mathbb{P}_0$, The initial polyhedron

Require: $A \in \mathbb{R}^n \times \mathbb{R}^m, b \in \mathbb{R}^m$

1: $D = \Delta$, $\Omega = \delta_{\mathbb{P}_0}(\Delta)$
2: **repeat**
3: $\quad \Omega' = \Omega$
4: $\quad$ **for all** $i = 0, \ldots, (l-1)$ **do**
5: $\quad\quad \Theta[i] = \Theta[i] + \langle b, D[i]\rangle$
6: $\quad\quad D[i] = A^T D[i]$
7: $\quad\quad \Upsilon[i] = \delta_{\mathbb{P}_0}(d[i]) + \Theta[i]$
8: $\quad\quad \Omega[i] = \max(\Omega[i], \Upsilon[i])$
9: $\quad$ **end for**
10: **until** $\Omega \sqsubseteq \Omega'$

$$\Omega_i(d) = \max\{\delta_{\mathbb{P}_0}(A^{Tj}d) + \sum_{k=1}^{j}\langle b, A^{T(k-1)}d\rangle, j = 0, .., i\}$$

Algorithm 2 Kleene Algorithm using support function.

Require: $\Delta \subset \mathbb{R}^n$, set of l directions. $\mathbb{P}_0$, The initial polyhedron

Require: $A \in \mathbb{R}^n \times \mathbb{R}^m, b \in \mathbb{R}^m$

1: $D = \Delta$, $\Omega = \delta_{\mathbb{P}_0}(\Delta)$
2: **repeat**
3: $\Omega' = \Omega$
4: **for all** $i = 0, \ldots, (l-1)$ **do**
5: $\Theta[i] = \Theta[i] + \langle b, D[i]\rangle$
6: $D[i] = A^T D[i]$
7: $\Upsilon[i] = \delta_{\mathbb{P}_0}(d[i]) + \Theta[i]$
8: $\Omega[i] = \max(\Omega[i], \Upsilon[i])$
9: **end for**
10: **until** $\Omega \sqsubseteq \Omega'$

$$\Omega_i(d) = \max\{\delta_{\mathbb{P}_0}(A^{Tj}d) + \sum_{k=1}^{j}\langle b, A^{T(k-1)}d\rangle, j = 0, .., i\}$$

Algorithm 3 Kleene Algorithm using support function.

Require: $\Delta \subset \mathbb{R}^n$, set of l directions. $\mathbb{P}_0$, The initial polyhedron

Require: $A \in \mathbb{R}^n \times \mathbb{R}^m, b \in \mathbb{R}^m$

1: $D = \Delta$, $\Omega = \delta_{\mathbb{P}_0}(\Delta)$
2: **repeat**
3: $\quad\Omega' = \Omega$
4: $\quad$**for all** $i = 0, \ldots, (l-1)$ **do**
5: $\quad\quad\Theta[i] = \Theta[i] + \langle b, D[i]\rangle$
6: $\quad\quad D[i] = A^T D[i]$
7: $\quad\quad\Upsilon[i] = \delta_{\mathbb{P}_0}(d[i]) + \Theta[i]$
8: $\quad\quad\Omega[i] = \max(\Omega[i], \Upsilon[i])$
9: $\quad$**end for**
10: **until** $\Omega \sqsubseteq \Omega'$

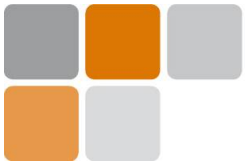
$$\Omega_i(d) = \max\{\delta_{\mathbb{P}_0}(A^{Tj}d) + \underbrace{\sum_{k=1}^{j}\langle b, A^{T(k-1)}d\rangle}, j = 0, .., i\}$$

Algorithm 4 Kleene Algorithm using support function.

Require: $\Delta \subset \mathbb{R}^n$, set of l directions. $\mathbb{P}_0$, The initial polyhedron

Require: $A \in \mathbb{R}^n \times \mathbb{R}^m, b \in \mathbb{R}^m$

1: $D = \Delta$, $\Omega = \delta_{\mathbb{P}_0}(\Delta)$
2: **repeat**
3: $\Omega' = \Omega$
4: **for all** $i = 0, \ldots, (l-1)$ **do**
5: $\Theta[i] = \Theta[i] + \langle b, D[i]\rangle$
6: $D[i] = A^T D[i]$
7: $\Upsilon[i] = \delta_{\mathbb{P}_0}(d[i]) + \Theta[i]$
8: $\Omega[i] = \max(\Omega[i], \Upsilon[i])$
9: **end for**
10: **until** $\Omega \sqsubseteq \Omega'$



The Algorithm doesn't guarantee the termination of the computation.

The Algorithm doesn't guarantee the termination of the computation.

- Solution 1 :

widening

$$\forall \Omega_1, \Omega_2 \in \mathbb{P}^{\sharp}_{\Delta},\ \Omega_1 \nabla_{\Delta} \Omega_2 = \lambda d. \begin{cases} \Omega_2(d) & \text{if } \Omega_1(d) = \Omega_2(d) \\ +\infty & \text{otherwise} \end{cases}$$

The Algorithm doesn't guarantee the termination of the computation.

- Solution 1 :

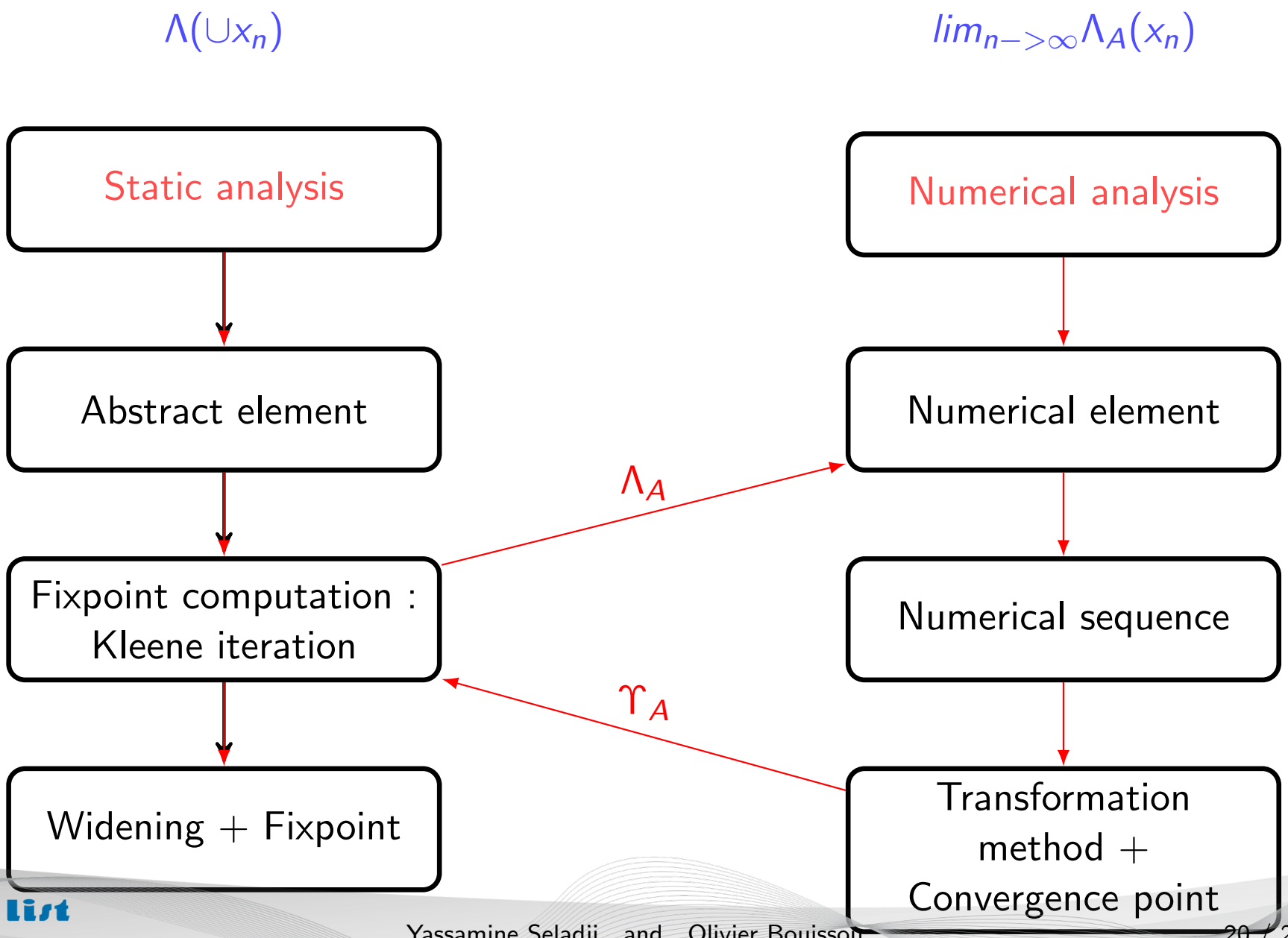
widening

$$\forall \Omega_1, \Omega_2 \in \mathbb{P}^{\sharp}_{\Delta},\ \Omega_1 \nabla_{\Delta} \Omega_2 = \lambda d. \begin{cases} \Omega_2(d) & \text{if } \Omega_1(d) = \Omega_2(d) \\ +\infty & \text{otherwise} \end{cases}$$

- Solution 2 :

The accelerated Kleene iteration

The accelerated Kleene iteration



The accelerated Kleene iteration

Algorithm 5 The accelerated Kleene iteration

1: **repeat**
2: $\quad \overrightarrow{X_i} := \overrightarrow{X_{i-1}} \sqcup F(\overrightarrow{X_{i-1}})$
3: $\quad \overrightarrow{y_i} := \text{Accelerate}\left(\Lambda_A(\overrightarrow{X_0}), \ldots, \Lambda_A(\overrightarrow{X_i})\right)$
4: $\quad$ **if** $||\overrightarrow{y_i} - \overrightarrow{y_{i-1}}|| \leq \delta$ **then**
5: $\quad\quad \overrightarrow{X_i} := \overrightarrow{X_i} \sqcup \Upsilon_A(\overrightarrow{y_i})$
6: $\quad$ **end if**
7: **until** $\overrightarrow{X_i} \sqsubseteq \overrightarrow{X_{i-1}}$

The accelerated Kleene iteration

- The method to accelerated the numerical sequences convergence.

Algorithm 6 The accelerated Kleene iteration

1: **repeat**
2: $\overrightarrow{X_i} := \overrightarrow{X_{i-1}} \sqcup F(\overrightarrow{X_{i-1}})$
3: $\overrightarrow{y_i} :=$ Accelerate $(\Lambda_A(X_0), \ldots, \Lambda_A(\overrightarrow{X_i}))$
4: **if** $||\overrightarrow{y_i} - \overrightarrow{y_{i-1}}|| \leq \delta$ **then**
5: $\overrightarrow{X_i} := \overrightarrow{X_i} \sqcup \Upsilon_A(\overrightarrow{y_i})$
6: **end if**
7: **until** $\overrightarrow{X_i} \sqsubseteq \overrightarrow{X_{i-1}}$

The accelerated Kleene iteration

- The method to accelerated the numerical sequences convergence.

Algorithm 7 The accelerated Kleene iteration

1: **repeat**
2: $\overrightarrow{X_i} := \overrightarrow{X_{i-1}} \sqcup F(\overrightarrow{X_{i-1}})$
3: $\overrightarrow{y_i} :=$ Accelerate $(\Lambda_A(X_0), \ldots, \Lambda_A(\overrightarrow{X_i}))$
4: **if** $||\overrightarrow{y_i} - \overrightarrow{y_{i-1}}|| \leq \delta$ **then**
5: $\overrightarrow{X_i} := \overrightarrow{X_i} \sqcup \Upsilon_A(\overrightarrow{y_i})$
6: **end if**
7: **until** $\overrightarrow{X_i} \sqsubseteq \overrightarrow{X_{i-1}}$

- The *Extraction* function

The accelerated Kleene iteration

- The method to accelerated the numerical sequences convergence.

Algorithm 8 The accelerated Kleene iteration

1: **repeat**
2: $\overrightarrow{X_i} := \overrightarrow{X_{i-1}} \sqcup F(\overrightarrow{X_{i-1}})$
3: $\overrightarrow{y_i} :=$ Accelerate $(\Lambda_A(X_0), \ldots, \Lambda_A(\overrightarrow{X_i}))$
4: **if** $||\overrightarrow{y_i} - \overrightarrow{y_{i-1}}|| \leq \delta$ **then**
5: $\overrightarrow{X_i} := \overrightarrow{X_i} \sqcup \Upsilon_A(\overrightarrow{y_i})$
6: **end if**
7: **until** $\overrightarrow{X_i} \sqsubseteq \overrightarrow{X_{i-1}}$

- The *Extraction* function
- The *Combination* function

The accelerated Kleene iteration

Example

```
while (1) {
xn1 = -0.4375 * x1 + 0.0625 * x2 + 0.2652 * x3 + 0.1 * u1 ;
xn2 = 0.0625 * x1 + 0.4375 * x2 + 0.2652 * x3 + 0.1 * u2 ;
xn3 = -0.2652 * x1 + 0.2652 * x2 + 0.375 * x3 + 0.1 * u3 ;
x1 = xn1 ;x2 = xn2 ;x3 = xn3 ;
} [-5.197505,8.873306]
```

Itération :3

x1	$\Upsilon_A(\overrightarrow{y})$	$\|y_i - y_{i-1}\|$	**Seuils** ($\delta = 10^{-3}$)
[1.000000,2.000000]	[1.000000,2.000000]		
[-0.447300,5.716500]			
[-2.291255,6.573381]			

The accelerated Kleene iteration

Example

```
while (1) {
xn1 = -0.4375 * x1 + 0.0625 * x2 + 0.2652 * x3 + 0.1 * u1 ;
xn2 = 0.0625 * x1 + 0.4375 * x2 + 0.2652 * x3 + 0.1 * u2 ;
xn3 = -0.2652 * x1 + 0.2652 * x2 + 0.375 * x3 + 0.1 * u3 ;
x1 = xn1 ;x2 = xn2 ;x3 = xn3 ;
} [-5.197505,8.873306]
```

Itération :3

x1	$\Upsilon_A(\overrightarrow{y})$	$\|y_i - y_{i-1}\|$	**Seuils** ($\delta = 10^{-3}$)
[1.000000,2.000000]	[1.000000,2.000000]		
[-0.447300,5.716500]			
[-2.291255,6.573381]	[6.280857,6.830145]	max(5.280857,4.830145)	

The accelerated Kleene iteration

Example

```
while (1) {
xn1 = -0.4375 * x1 + 0.0625 * x2 + 0.2652 * x3 + 0.1 * u1 ;
xn2 = 0.0625 * x1 + 0.4375 * x2 + 0.2652 * x3 + 0.1 * u2 ;
xn3 = -0.2652 * x1 + 0.2652 * x2 + 0.375 * x3 + 0.1 * u3 ;
x1 = xn1 ;x2 = xn2 ;x3 = xn3 ;
} [-5.197505,8.873306]
```

Itération :5

x1	$\Upsilon_A(\overrightarrow{y})$	$\|y_i - y_{i-1}\|$	**Seuils** ($\delta = 10^{-3}$)
[1.000000,2.000000]	[1.000000,2.000000]		
[-0.447300,5.716500]			
[-2.291255,6.573381]	[6.280857,6.830145]	max(5.280857,4.830145)	
[-3.038029,7.492492]			
[-3.695558,7.871720]	[-4.663282,8.463092]	max(10.944139,2.367053)	

The accelerated Kleene iteration

Example

```
while (1) {
xn1 = -0.4375 * x1 + 0.0625 * x2 + 0.2652 * x3 + 0.1 * u1 ;
xn2 = 0.0625 * x1 + 0.4375 * x2 + 0.2652 * x3 + 0.1 * u2 ;
xn3 = -0.2652 * x1 + 0.2652 * x2 + 0.375 * x3 + 0.1 * u3 ;
x1 = xn1 ;x2 = xn2 ;x3 = xn3 ;
} [-5.197505,8.873306]
```

Itération :7

x1	$\Upsilon_A(\overrightarrow{y})$	$\|y_i - y_{i-1}\|$	**Seuils** ($\delta = 10^{-3}$)
[1.000000,2.000000]	[1.000000,2.000000]		
[-0.447300,5.716500]			
[-2.291255,6.573381]	[6.280857,6.830145]	max(5.280857,4.830145)	
[-3.038029,7.492492]			
[-3.695558,7.871720]	[-4.663282,8.463092]	max(10.944139,2.367053)	
[-4.084503,8.185954]			
[-4.386442,8.369221]	[-5.189239,8.851176]	max(0.525957,0.388084)	

The accelerated Kleene iteration

Example

```
while (1) {
xn1 = -0.4375 * x1 + 0.0625 * x2 + 0.2652 * x3 + 0.1 * u1 ;
xn2 = 0.0625 * x1 + 0.4375 * x2 + 0.2652 * x3 + 0.1 * u2 ;
xn3 = -0.2652 * x1 + 0.2652 * x2 + 0.375 * x3 + 0.1 * u3 ;
x1 = xn1 ;x2 = xn2 ;x3 = xn3 ;
} [-5.197505,8.873306]
```

Itération :9

x1	$\Upsilon_A(\overrightarrow{y})$	$\|y_i - y_{i-1}\|$	**Seuils** ($\delta = 10^{-3}$)
[1.000000,2.000000]	[1.000000,2.000000]		
[-0.447300,5.716500]			
[-2.291255,6.573381]	[6.280857,6.830145]	max(5.280857,4.830145)	
[-3.038029,7.492492]			
[-3.695558,7.871720]	[-4.663282,8.463092]	max(10.944139,2.367053)	
[-4.084503,8.185954]			
[-4.386442,8.369221]	[-5.189239,8.851176]	max(0.525957,0.388084)	
[-4.594886,8.508279]			
[-4.751445,8.603235]	[-5.197089,8.872567]	max(0.00785,0.021391)	

The accelerated Kleene iteration

Example

```
while (1) {
xn1 = -0.4375 * x1 + 0.0625 * x2 + 0.2652 * x3 + 0.1 * u1 ;
xn2 = 0.0625 * x1 + 0.4375 * x2 + 0.2652 * x3 + 0.1 * u2 ;
xn3 = -0.2652 * x1 + 0.2652 * x2 + 0.375 * x3 + 0.1 * u3 ;
x1 = xn1 ;x2 = xn2 ;x3 = xn3 ;
} [-5.197505,8.873306]
```

Itération :11

x1	$\Upsilon_A(\overrightarrow{y})$	$\|y_i - y_{i-1}\|$	**Seuils** ($\delta = 10^{-3}$)
[1.000000,2.000000]	[1.000000,2.000000]		
[-0.447300,5.716500]			
[-2.291255,6.573381]	[6.280857,6.830145]	max(5.280857,4.830145)	
[-3.038029,7.492492]			
[-3.695558,7.871720]	[-4.663282,8.463092]	max(10.944139,2.367053)	
[-4.084503,8.185954]			
[-4.386442,8.369221]	[-5.189239,8.851176]	max(0.525957,0.388084)	
[-4.594886,8.508279]			
[-4.751445,8.603235]	[-5.197089,8.872567]	max(0.00785,0.021391)	
[-4.865385,8.673918]			
[-4.950393,8.725095]	[-5.197506,8.873307]	max(0.000417,0.00074)	[-5.197506,8.873307]

The accelerated Kleene iteration

Example

```
while (1) {
xn1 = -0.4375 * x1 + 0.0625 * x2 + 0.2652 * x3 + 0.1 * u1 ;
xn2 = 0.0625 * x1 + 0.4375 * x2 + 0.2652 * x3 + 0.1 * u2 ;
xn3 = -0.2652 * x1 + 0.2652 * x2 + 0.375 * x3 + 0.1 * u3 ;
x1 = xn1 ;x2 = xn2 ;x3 = xn3 ;
} [-5.197505,8.873306]
```

Itération : 12

x1	$\Upsilon_A(\overrightarrow{y})$	$\|y_i - y_{i-1}\|$	**Seuils** ($\delta = 10^{-3}$)
[1.000000,2.000000]	[1.000000,2.000000]		
[-0.447300,5.716500]			
[-2.291255,6.573381]	[6.280857,6.830145]	max(5.280857,4.830145)	
[-3.038029,7.492492]			
[-3.695558,7.871720]	[-4.663282,8.463092]	max(10.944139,2.367053)	
[-4.084503,8.185954]			
[-4.386442,8.369221]	[-5.189239,8.851176]	max(0.525957,0.388084)	
[-4.594886,8.508279]			
[-4.751445,8.603235]	[-5.197089,8.872567]	max(0.00785,0.021391)	
[-4.865385,8.673918]			
[-4.950393,8.725095]	[-5.197506,8.873307]	max(0.000417,0.00074)	[-5.197506,8.873307]
[-5.197506,8.873307]			

The accelerated Kleene iteration

Example

```
while (1) {
xn1 = -0.4375 * x1 + 0.0625 * x2 + 0.2652 * x3 + 0.1 * u1 ;
xn2 = 0.0625 * x1 + 0.4375 * x2 + 0.2652 * x3 + 0.1 * u2 ;
xn3 = -0.2652 * x1 + 0.2652 * x2 + 0.375 * x3 + 0.1 * u3 ;
x1 = xn1 ;x2 = xn2 ;x3 = xn3 ;
} [-5.197505,8.873306]
```

Itération : 13

x1	$\Upsilon_A(\overrightarrow{y})$	$\|y_i - y_{i-1}\|$	**Seuils** ($\delta = 10^{-3}$)
[1.000000,2.000000]	[1.000000,2.000000]		
[-0.447300,5.716500]			
[-2.291255,6.573381]	[6.280857,6.830145]	max(5.280857,4.830145)	
[-3.038029,7.492492]			
[-3.695558,7.871720]	[-4.663282,8.463092]	max(10.944139,2.367053)	
[-4.084503,8.185954]			
[-4.386442,8.369221]	[-5.189239,8.851176]	max(0.525957,0.388084)	
[-4.594886,8.508279]			
[-4.751445,8.603235]	[-5.197089,8.872567]	max(0.00785,0.021391)	
[-4.865385,8.673918]			
[-4.950393,8.725095]	[-5.197506,8.873307]	max(0.000417,0.00074)	[-5.197506,8.873307]
[-5.197506,8.873307]			
[-5.197506,8.873307]	[-5.197089,8.872567]	max(0.000417,0.00074)	[-5.197089,8.872567]

The accelerated Kleene iteration

Example

```
while (1) {
xn1 = -0.4375 * x1 + 0.0625 * x2 + 0.2652 * x3 + 0.1 * u1 ;
xn2 = 0.0625 * x1 + 0.4375 * x2 + 0.2652 * x3 + 0.1 * u2 ;
xn3 = -0.2652 * x1 + 0.2652 * x2 + 0.375 * x3 + 0.1 * u3 ;
x1 = xn1 ;x2 = xn2 ;x3 = xn3 ;
} [-5.197505,8.873306]
```

Itération : 15

x1	$\Upsilon_A(\overrightarrow{y})$	$\|y_i - y_{i-1}\|$	**Seuils** ($\delta = 10^{-3}$)
[1.000000,2.000000]	[1.000000,2.000000]		
[-0.447300,5.716500]			
[-2.291255,6.573381]	[6.280857,6.830145]	max(5.280857,4.830145)	
[-3.038029,7.492492]			
[-3.695558,7.871720]	[-4.663282,8.463092]	max(10.944139,2.367053)	
[-4.084503,8.185954]			
[-4.386442,8.369221]	[-5.189239,8.851176]	max(0.525957,0.388084)	
[-4.594886,8.508279]			
[-4.751445,8.603235]	[-5.197089,8.872567]	max(0.00785,0.021391)	
[-4.865385,8.673918]			
[-4.950393,8.725095]	[-5.197506,8.873307]	max(0.000417,0.00074)	[-5.197506,8.873307]
[-5.197506,8.873307]			
[-5.197506,8.873307]	[-5.197089,8.872567]	max(0.000417,0.00074)	[-5.197089,8.872567]
[-5.197567,8.873341]			
[-5.197616,8.873348]	[-5.198638,8.849716]	max(0.001549,0.022851)	

The accelerated Kleene iteration

Example

```
while (1) {
xn1 = -0.4375 * x1 + 0.0625 * x2 + 0.2652 * x3 + 0.1 * u1 ;
xn2 = 0.0625 * x1 + 0.4375 * x2 + 0.2652 * x3 + 0.1 * u2 ;
xn3 = -0.2652 * x1 + 0.2652 * x2 + 0.375 * x3 + 0.1 * u3 ;
x1 = xn1 ;x2 = xn2 ;x3 = xn3 ;
} [-5.197505,8.873306] ↷ [-5.197598,8.873362]
```

Itération : 16

x1	$\Upsilon_A(\overrightarrow{y})$	$\|y_i - y_{i-1}\|$	Seuils ($\delta = 10^{-3}$)
[1.000000,2.000000]	[1.000000,2.000000]		
[-0.447300,5.716500]			
[-2.291255,6.573381]	[6.280857,6.830145]	max(5.280857,4.830145)	
[-3.038029,7.492492]			
[-3.695558,7.871720]	[-4.663282,8.463092]	max(10.944139,2.367053)	
[-4.084503,8.185954]			
[-4.386442,8.369221]	[-5.189239,8.851176]	max(0.525957,0.388084)	
[-4.594886,8.508279]			
[-4.751445,8.603235]	[-5.197089,8.872567]	max(0.00785,0.021391)	
[-4.865385,8.673918]			
[-4.950393,8.725095]	[-5.197506,8.873307]	max(0.000417,0.00074)	[-5.197506,8.873307]
[-5.197506,8.873307]			
[-5.197506,8.873307]	[-5.197089,8.872567]	max(0.000417,0.00074)	[-5.197089,8.872567]
[-5.197567,8.873341]			
[-5.197616,8.873348]	[-5.198638,8.849716]	max(0.001549,0.022851)	
[-5.197598,8.873362]			

Accelerated Kleene Algorithm using support function

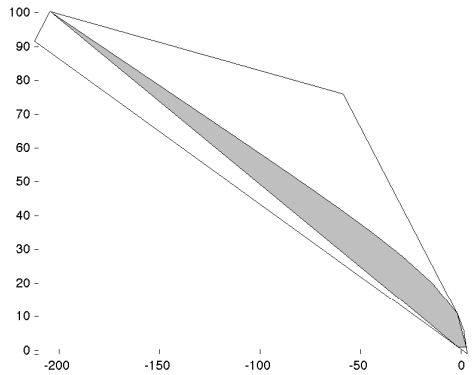
Algorithm 9 Accelerated Kleene Algorithm using support function.

Require: $\Delta \subset \mathbb{R}^n$, $\mathbb{P}_0$, $A \in \mathbb{R}^n \times \mathbb{R}^m$, $b \in \mathbb{R}^m$

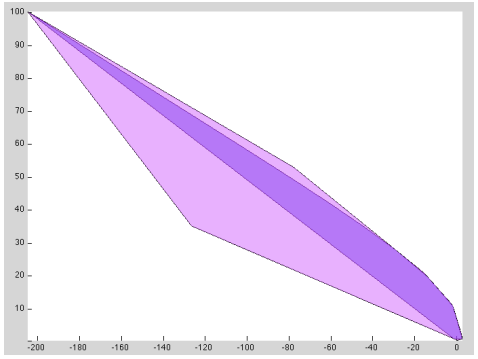
$D = \Delta$, $\Omega = \delta_{\mathbb{P}_0}(\Delta)$
repeat
  $\Omega' = \Omega$, $Y' = Y$
  for all $i = 0, \ldots, (l-1)$ **do**
    $\Theta[i] = \Theta[i] + \langle b, D[i] \rangle$
    $D[i] = A^T D[i]$
    $\Upsilon[i] = \delta_{\mathbb{P}_0}(d[i]) + \Theta[i]$
    $Y[i] :=$ Accelerate $(\Upsilon[0], \ldots, \Upsilon[i])$
    if $||Y[i] - Y'[i]|| \leq \varepsilon$ **then**
      $\Omega[i] = \max(\Omega[i], Y[i])$
    else
      $\Omega[i] = \max(\Omega[i], \Upsilon[i])$
    end if
  end for
until $\Omega \sqsubseteq \Omega'$

Kleene Algorithm using support function

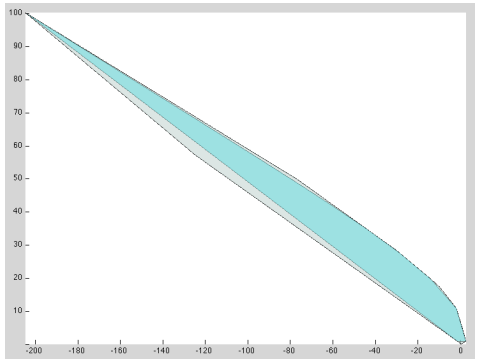
```
begin
    while (0<=10) do
    xn = 0.5 *x - y - 2.5;
     yn = 0.9 *y + 10;
     x = xn; y = yn;
    done;
end
```



8 directions
(0.044 seconde)



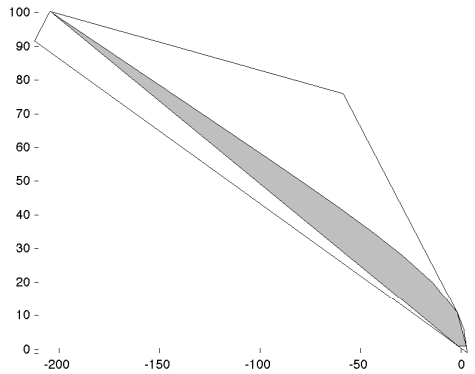
50 directions
(0.34 seconde)



100 directions
(0.7 seconde)

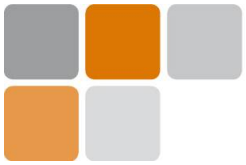
Accelerated Kleene Algorithm using support function

```
begin
    while (0<=10) do
    xn = 0.5 *x - y - 2.5;
     yn = 0.9 *y + 10;
     x = xn; y = yn;
    done;
end
```



8 directions

- Kleene iteration using support function : 200 iterations
- Accelerated Kleene iteration using support function : 11 iterations



S. Sankaranarayanan, H. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In VMCAI. Springer, 2005.

S. Sankaranarayanan, H. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In VMCAI. Springer, 2005.

Similarities :

- Abstract domain based on a static choice of directions set.
- The same definition of inclusion, meet and join operators.

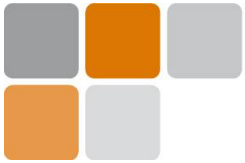
S. Sankaranarayanan, H. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In VMCAI. Springer, 2005.

Similarities :

- Abstract domain based on a static choice of directions set.
- The same definition of inclusion, meet and join operators.

Differences :

- The fixpoint computation uses Linear Programming.
- The fixpoint is computed with the abstract elements obtained in each Kleene iteration.



Conclusion and future work

Conclusion

- We develop a new numerical abstract domain based on support function.
- Our abstract domain depends on a set of finite directions.
- The fixpoint is computed in a polynomial time.

Perspectives

- Implements this domain on APRON.
- According to the program to analyse, defines a relevant set of directions.