Détection de *heap overflow* par analyse statique

Xavier Allamigeon[1,2] Charles Hymans[1]

[1]EADS Innovation Works, SE/CS – Suresnes

[2]CEA-LIST MeASI – Gif-sur-Yvette

firstname.lastname@eads.net

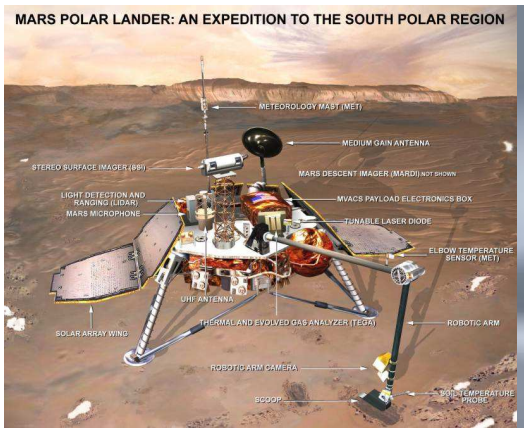
31 mai 2007

Le logiciel aujourd'hui

Situation actuelle du logiciel : présent dans tous les domaines.

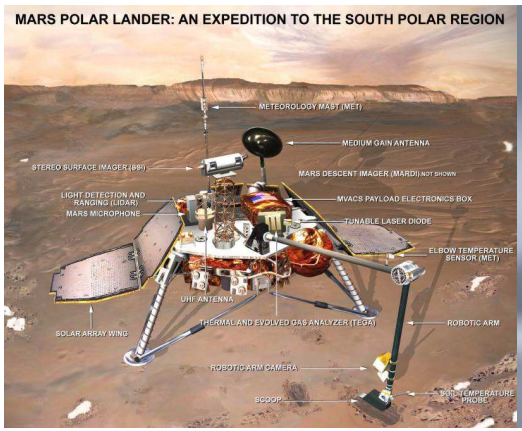
Le logiciel aujourd'hui

Situation actuelle du logiciel : présent dans tous les domaines.



Le logiciel aujourd'hui

Situation actuelle du logiciel : présent dans tous les domaines.

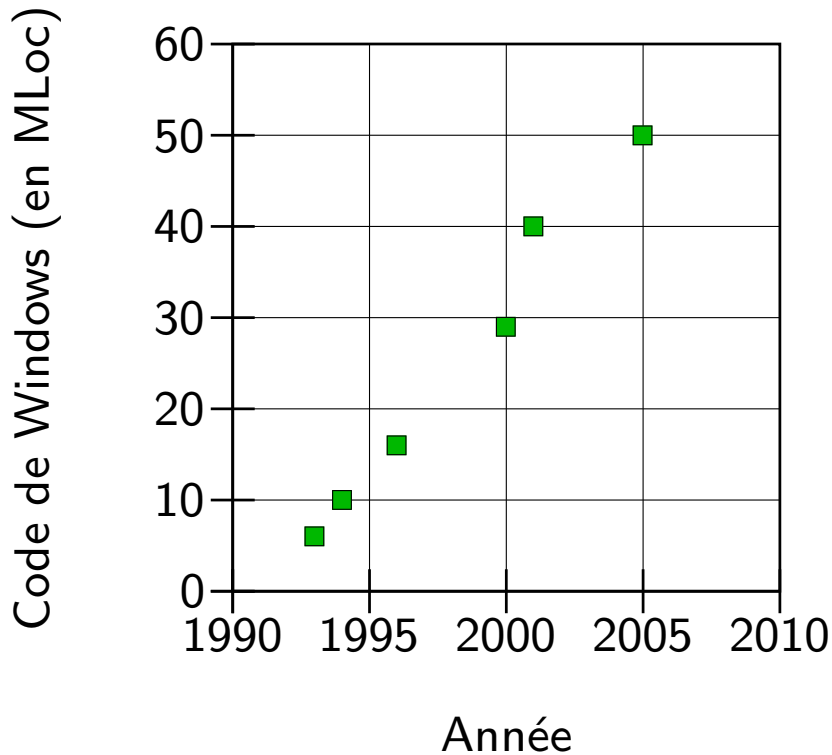


. . . et de plus en plus critiques !

Le logiciel aujourd'hui (2)

Croissance de la taille du code :

Produit	Code (en MLoc)
Télévision	1
Graveur DVD	2.5
Boeing 777	4
Voiture	35
Windows Vista	50



Les bugs dans les logiciels [4]

- avant relecture ou test, entre 0.5 et 5 erreurs pour 100 lignes de code de C.
- relecture : une équipe de 4 ingénieurs examine 150 lignes de code par heure, et découvre 70% des bugs.
  Exemple : pour un logiciel d'un million de lignes
  - 50 000 bugs au début.
  - la relecture dure 14 ans.
  - il reste encore 15 000 bugs.
- test : ne couvre pas tous les comportements possibles (50%).

Aucune certitude sur l'absence de bugs, même dans des codes critiques !

Montrer l'absence de bugs

On utilise pour cela *l'analyse statique par interprétation abstraite*.

- *Analyse statique* : sans exécuter le logiciel, détecter les erreurs qu'il pourrait provoquer.
- *Interprétation abstraite* [2] :
  - théorie permettant de sur-approximer l'ensemble de *tous* les comportements possibles d'un programme.
  - les analyses sont *correctes* (ou *sûres*) : elles montrent l'absence de bugs.
  - contre-partie : fausses alarmes possibles.

Enjeu : construire une analyse précise, entièrement automatique et performante.

Plan de l'exposé

Objectif : analyse statique par interprétation abstraite pour montrer l'absence de dépassements de tampon dans le tas (*heap overflow*).

Langage analysé

Sémantique du langage

- Modélisation des états de la mémoire
- Sémantique des instructions
- Sémantique collectrice

Abstraction

- Approximations par polyèdres convexes
- Le formalisme
- Abstraction des états de la machine
- Algorithme de l'analyse

Extensions possibles et travaux relatifs

Exemple : programme collectant des données de l'extérieur

```
unsigned char* receive(unsigned int n) {
    assert (n>0);
    unsigned char* t = (unsigned char*)malloc(n);
    readbuf(t,n);
    return t;
}

void readbuf(unsigned char* t, unsigned int n) {
    unsigned int i, sz;
    sz = (unsigned int) getchar();
    if (sz > n)
       sz = n;
    for (i = 0; i < sz; i++)
       *(t+i) = (unsigned char) getchar();
}
```

Exemple : programme collectant des données de l'extérieur

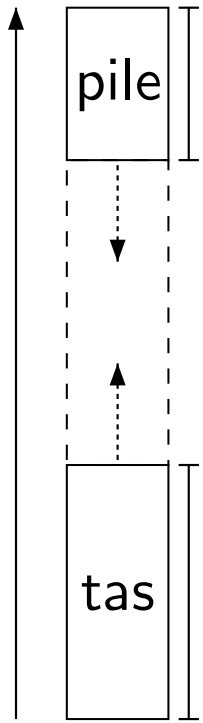
```
unsigned char* receive(unsigned int n) {
    assert (n>0);
    unsigned char* t = (unsigned char*)malloc(n);
    readbuf(t,n);
    return t;
}

void readbuf(unsigned char* t, unsigned int n) {
    unsigned int i, sz;
    sz = (unsigned int) getchar();
    if (sz > n)
       sz = n;
    for (i = 0; i < sz; i++)
       *(t+i) = (unsigned char) getchar();
}
```

Plan de l'exposé

Langage analysé

Forme « noyau » pour C :

- les types sont **unsigned char** (**uchar**) et **uchar**$*$.
- la mémoire consiste en deux parties disjointes :



variables locales allouées statiquement

mémoire allouée dynamiquement (malloc)
ne contient que des **uchar**

- pas de **null**, pas de &.
- seules fonctions autorisées : malloc et getuchar

Langage analysé (2)

Tous les malloc d'un programme sont étiquetés par des symboles distincts α, β, ..., appelés *sites d'allocation* (Alloc).

Exemple

$$
\begin{array}{ll}
1: & t = \mathrm{malloc}_{\alpha}(n); \\
2: & i = 0; \\
3: & p = \mathrm{malloc}_{\beta}(n-1); \\
 & \vdots
\end{array}
$$

Traduction possible de notre exemple

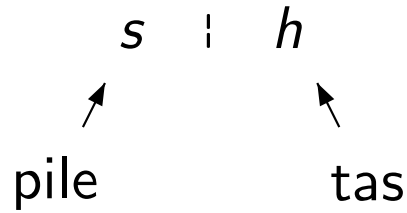
$$
\begin{array}{rl}
1: & t = \mathrm{malloc}_{\alpha}(n); \\
2: & sz = \mathrm{getuchar}(); \\
3: & \mathrm{if}\ (!(sz - n \leq 0)) \\
4: & \quad sz = n; \\
5: & i = 0; \\
6: & p = t; \\
7: & \mathrm{while}\ (i - sz + 1 \leq 0)\ \{ \\
8: & \quad tmp = \mathrm{getuchar}(); \\
9: & \quad *p = tmp; \\
10: & \quad p = p + 1; \\
11: & \quad i = i + 1; \\
12: & \}
\end{array}
$$

Plan de l'exposé

Plan de l'exposé

Modélisation des états de la mémoire

Un état de la mémoire :



s associe :

- à chaque variable de type **uchar**, un entier $n \in \mathbb{N}$.
- à chaque pointeur,
  - ω s'il n'est pas initialisé.
  - une adresse a en mémoire sinon.

Exemple

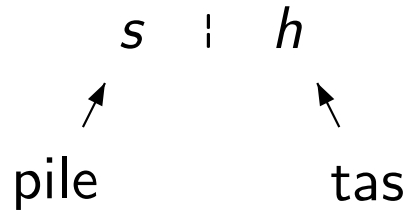
Si i et n sont de type **uchar** et p de type **uchar**$*$:

$$\begin{array}{ll} 1: & n = 10; \\ 2: & i = 5; \end{array}$$

$$s_{final} : \begin{cases} i \mapsto 5 \\ n \mapsto 10 \\ p \mapsto \omega \end{cases}$$

Modélisation des états de la mémoire

Un état de la mémoire :



s associe :

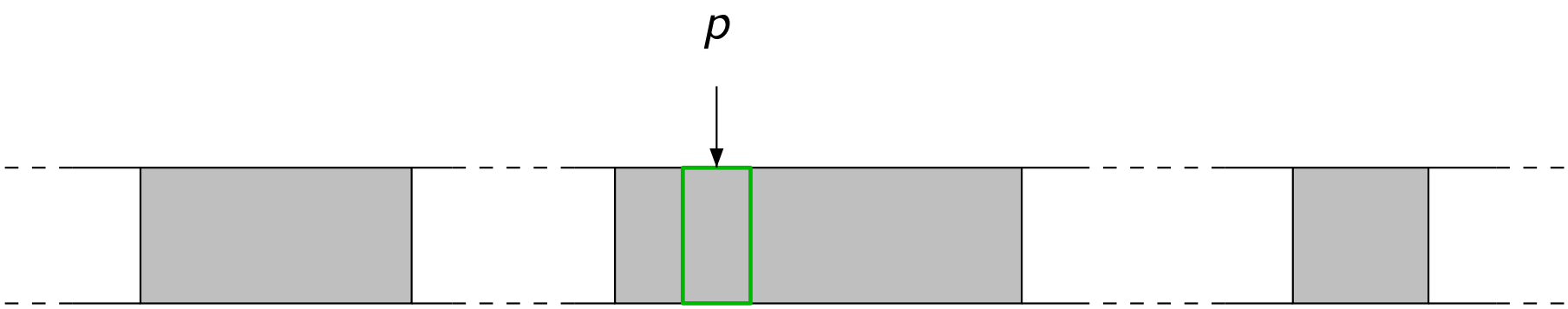
- à chaque variable de type **uchar**, un entier $n \in \mathbb{N}$.
- à chaque pointeur,
  - ω s'il n'est pas initialisé.
  - une adresse a en mémoire sinon.

h associe les adresses allouées dynamiquement à des caractères.

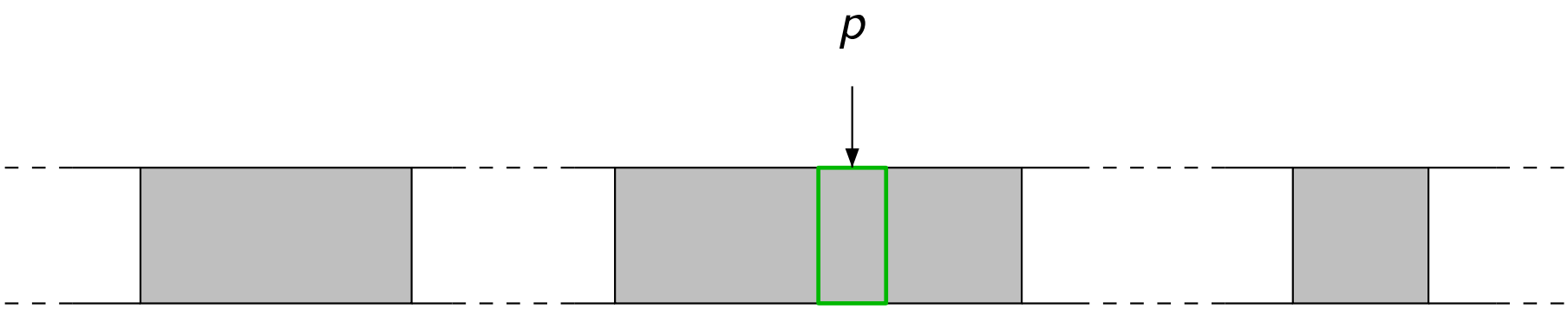
Modélisation du tas

Modélisation du tas

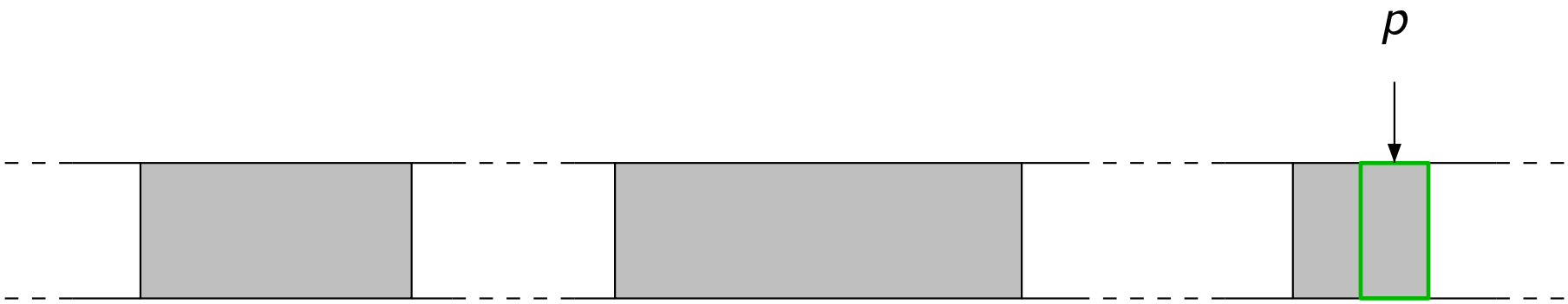
Modélisation du tas



Modélisation du tas



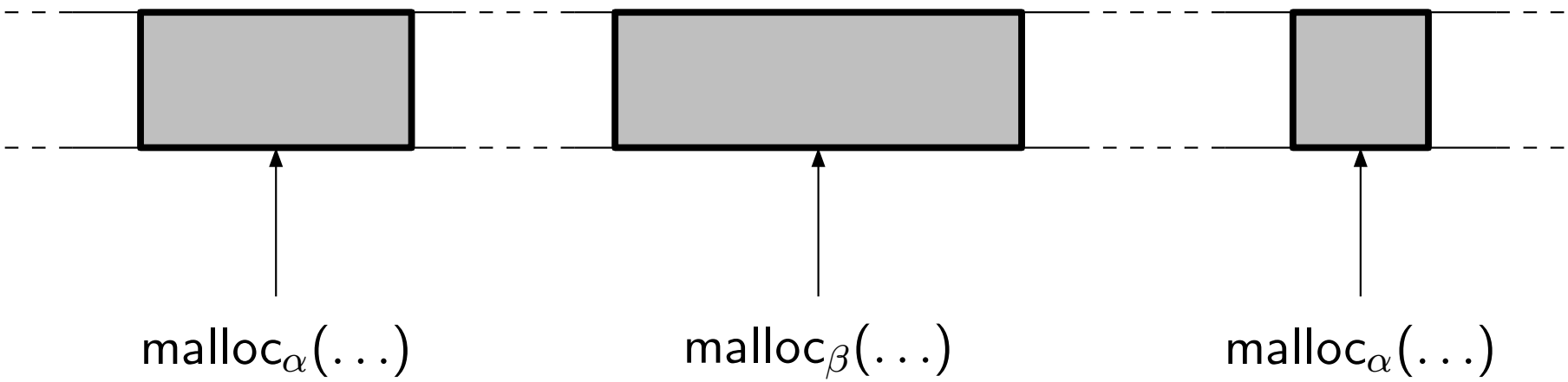
Modélisation du tas



Modélisation du tas

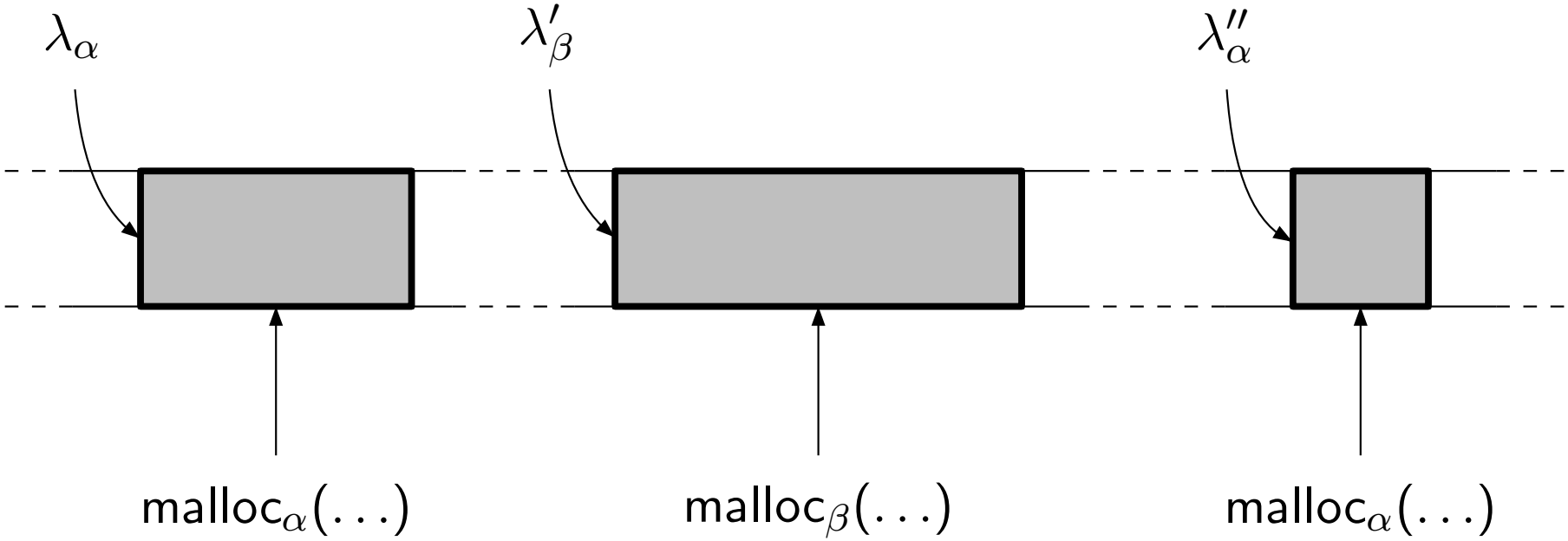
Modélisation du tas

Modélisation du tas



$$
\begin{array}{rl}
 & \vdots \\
4: & \quad t = \mathrm{malloc}_\beta(n); \\
 & \vdots \\
11: & \quad \text{while } \ldots \{ \\
12: & \quad\quad p = \mathrm{malloc}_\alpha(i+1); \\
 & \vdots
\end{array}
$$

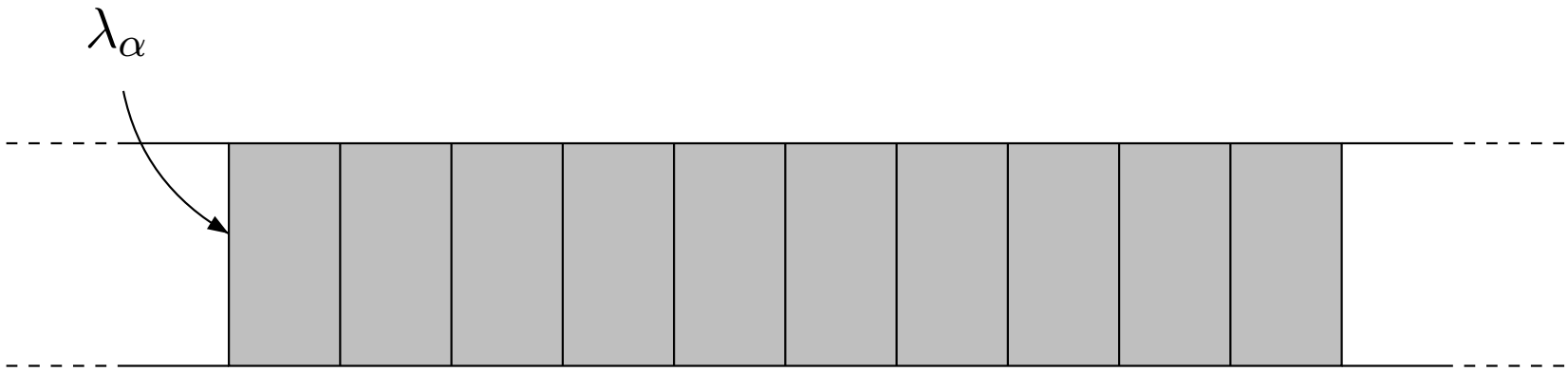
Modélisation du tas



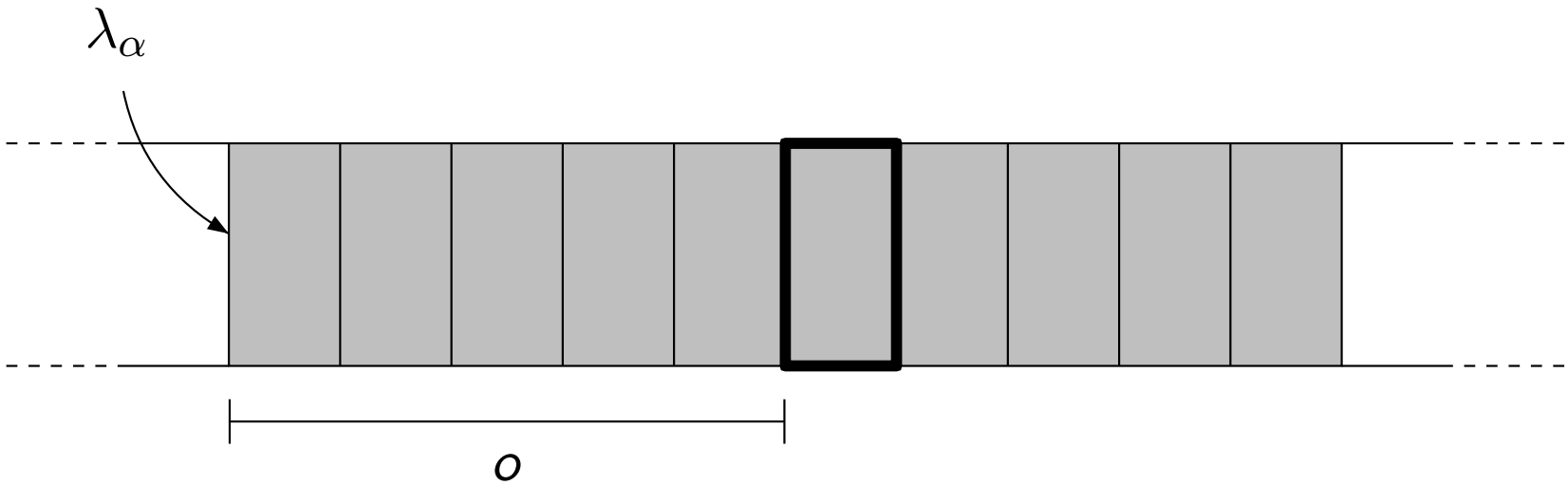
$$
\begin{array}{rl}
 & \vdots \\
4: & \quad t = \text{malloc}_\beta(n); \\
 & \vdots \\
11: & \quad \text{while } \ldots \{ \\
12: & \quad\quad p = \text{malloc}_\alpha(i+1); \\
 & \vdots
\end{array}
$$

Modélisation du tas (2)

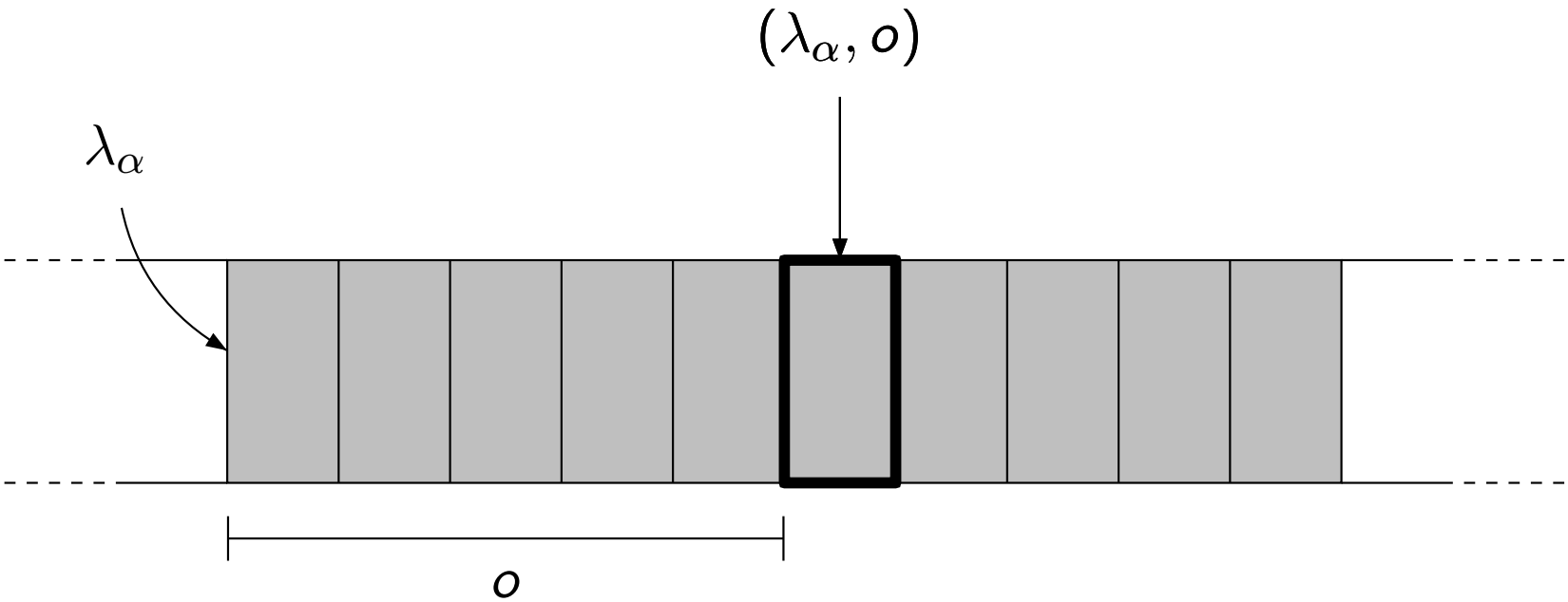
Modélisation du tas (2)



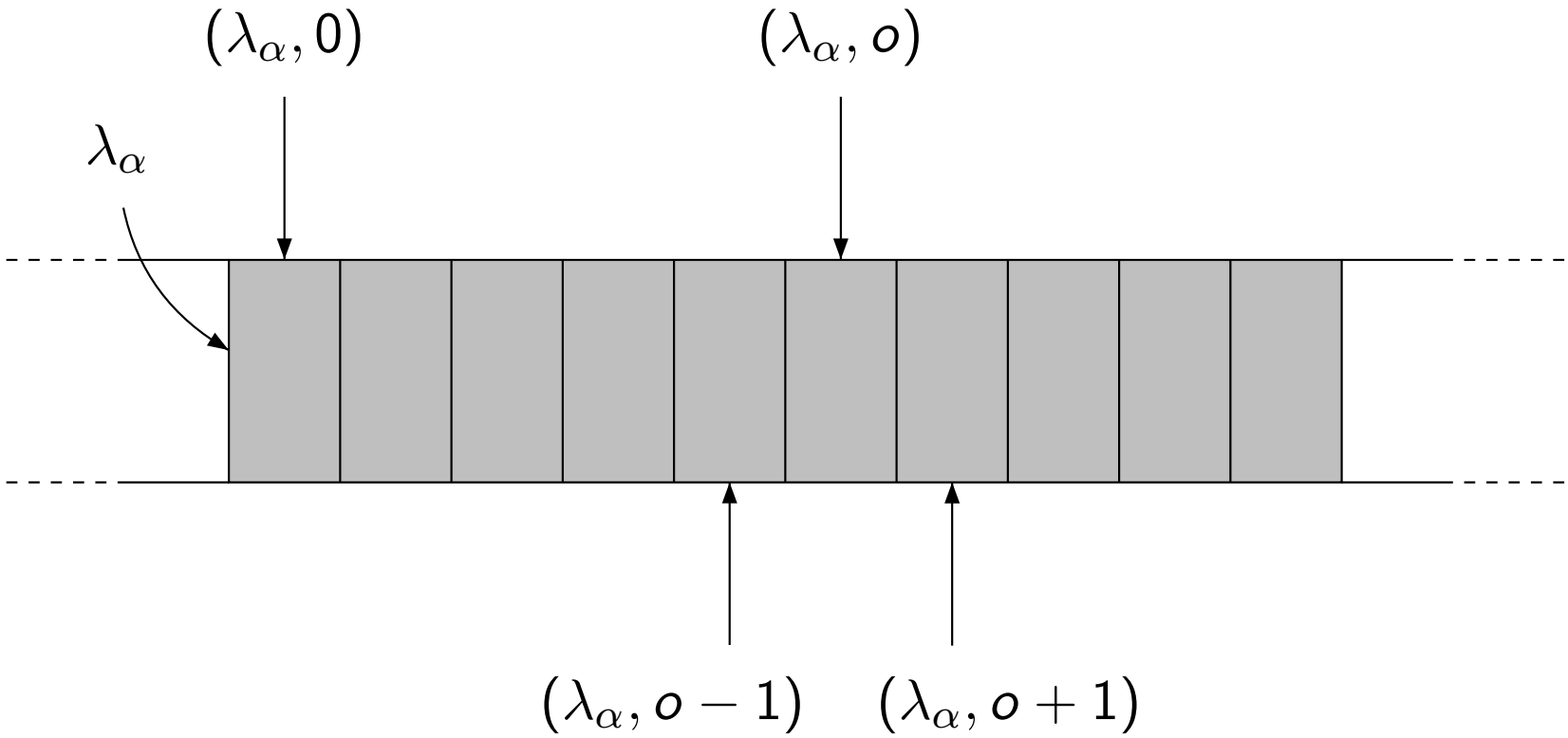
Modélisation du tas (2)



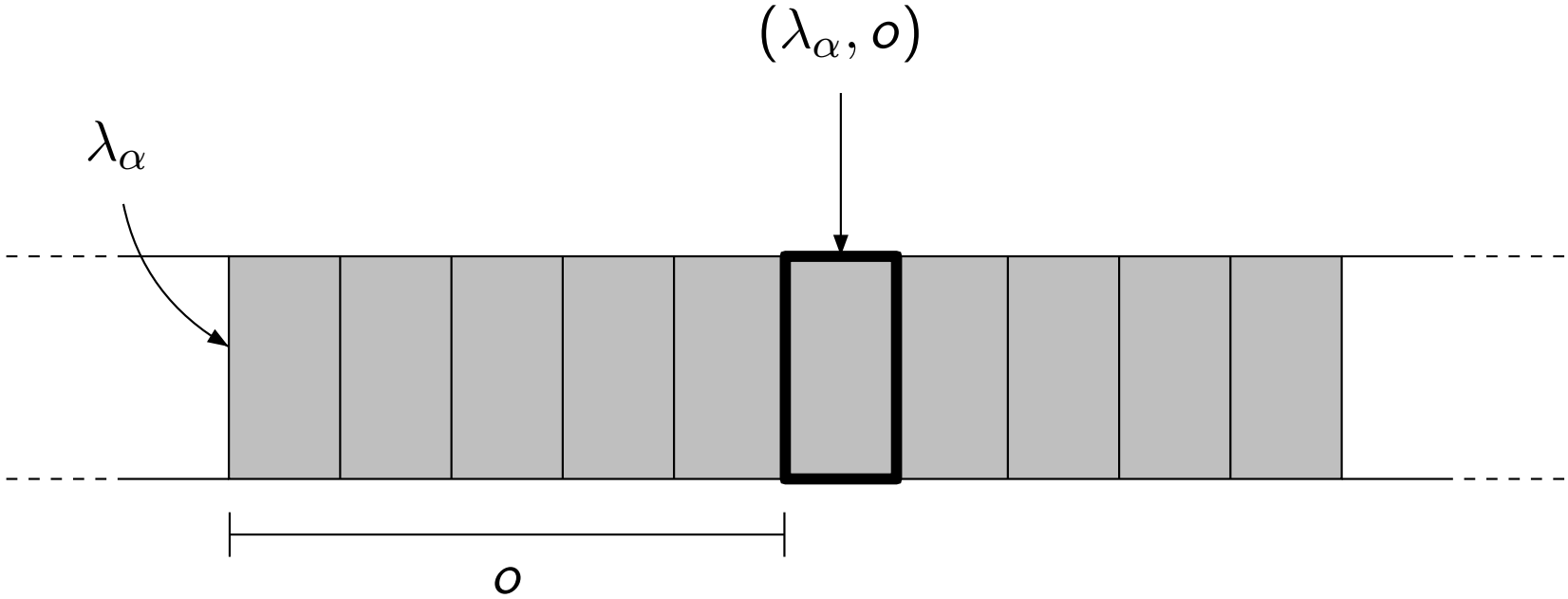
Modélisation du tas (2)



Modélisation du tas (2)



Modélisation du tas (2)



adresse : $a = (\lambda_\alpha, o) \in \text{Addr}$

- λ_α : *location* du bloc dans lequel se situe l'adresse.
- o : *décalage* (*offset*) par rapport au début du bloc.

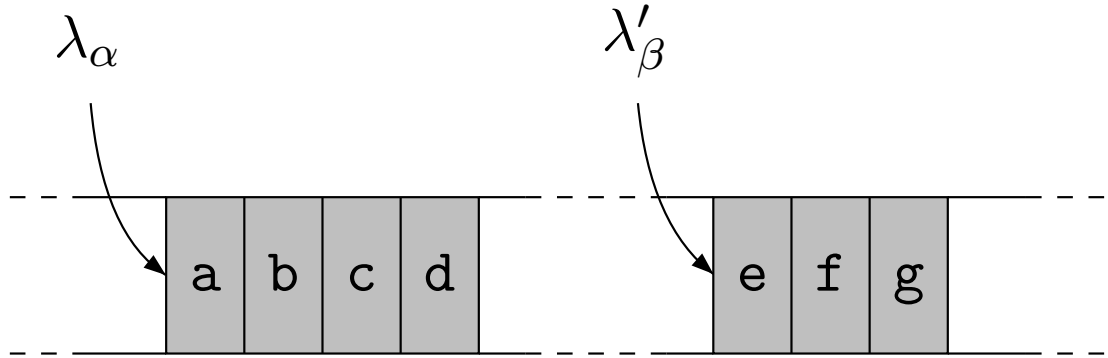
Modélisation du tas (3)

h est une fonction *partiellement définie* sur Addr, et à valeurs dans $\mathbb{N}$ (caractères).

$$h : \text{Addr} \rightharpoonup \mathbb{N}$$

Son ensemble de définition $\text{dom}(h)$ est exactement l'ensemble des adresses allouées dans le tas.

Exemple



h est définie sur $\{(\lambda_\alpha, 0), \ldots, (\lambda_\alpha, 3), (\lambda'_\beta, 0), \ldots, (\lambda'_\beta, 2)\}$, par :

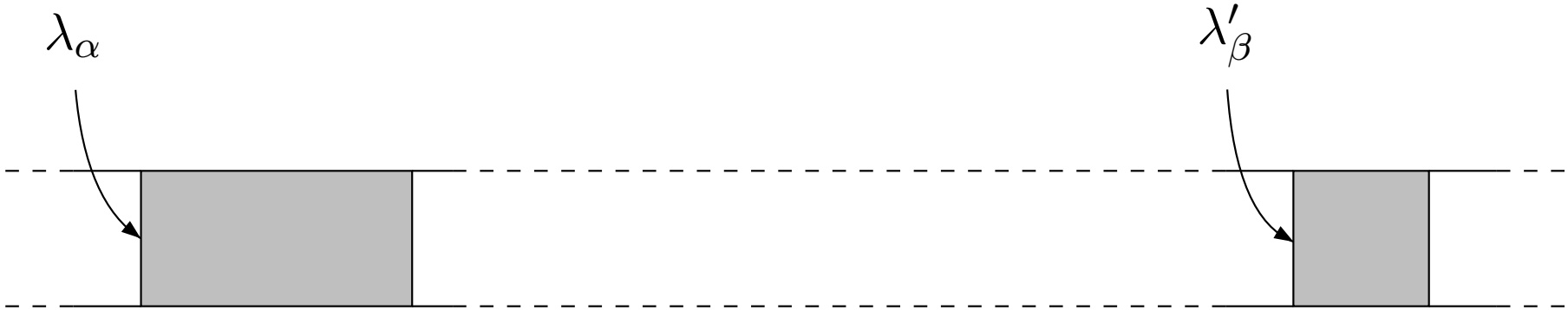
$$h : \begin{cases} (\lambda_\alpha, 0) \mapsto 97 & (\lambda_\alpha, 1) \mapsto 98 & (\lambda_\alpha, 2) \mapsto 99 & (\lambda_\alpha, 3) \mapsto 100 \\ (\lambda'_\beta, 0) \mapsto 101 & (\lambda'_\beta, 1) \mapsto 102 & (\lambda'_\beta, 2) \mapsto 103 & \end{cases}$$

Plan de l'exposé

Sémantique des instructions

Sémantique d'une instruction : quel est l'état $s' \restriction h'$ de la mémoire après l'exécution de *instr* sur l'état $s \restriction h$?

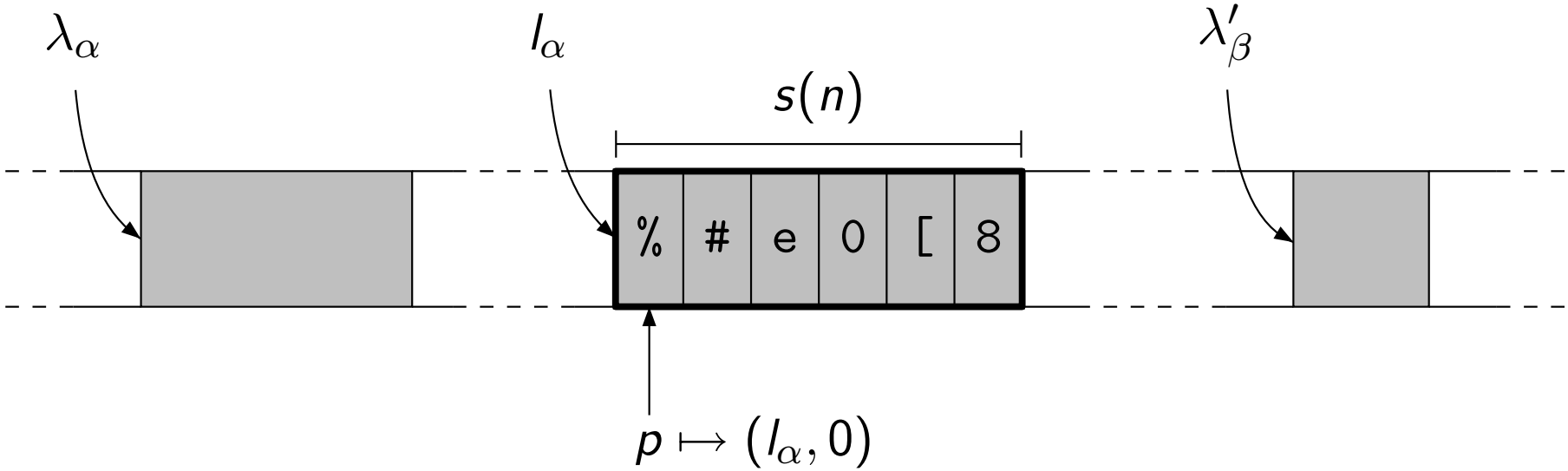
- pour $p = \text{malloc}_\alpha(n)$: *avant*



Sémantique des instructions

Sémantique d'une instruction : quel est l'état $s' \restriction h'$ de la mémoire après l'exécution de *instr* sur l'état $s \restriction h$?

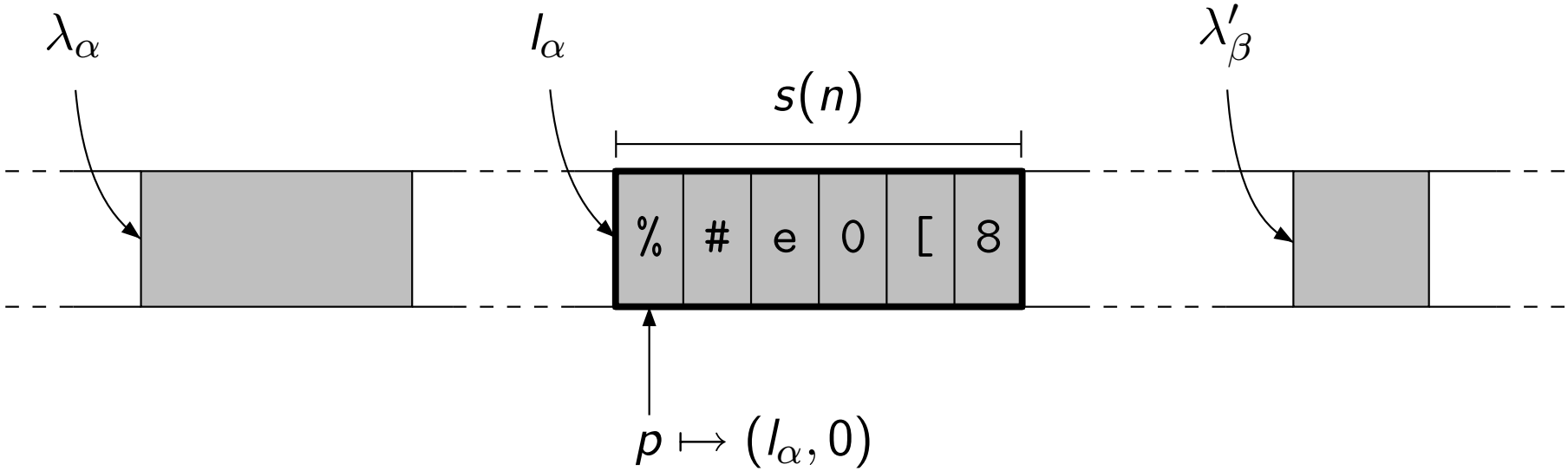
- pour $p = \text{malloc}_\alpha(n)$: *après*



Sémantique des instructions

Sémantique d'une instruction : quel est l'état $s' \dagger h'$ de la mémoire après l'exécution de *instr* sur l'état $s \dagger h$?

- pour $p = \text{malloc}_\alpha(n)$:



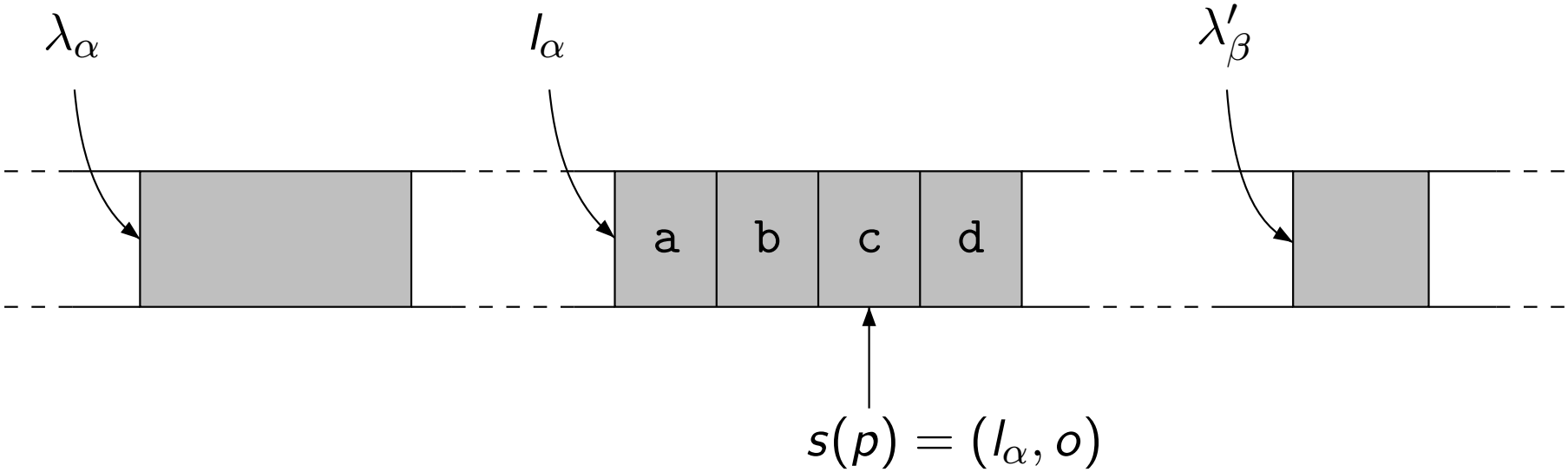
Sous forme mathématique,

$$s' = s[p \mapsto (l_\alpha, 0)]$$
$$h' = h[(l_\alpha, 0) \mapsto w_0] \ldots [(l_\alpha, k-1) \mapsto w_{k-1}]$$

avec $s(n) = k$ et l_α n'apparaît pas dans $\text{dom}(h)$.

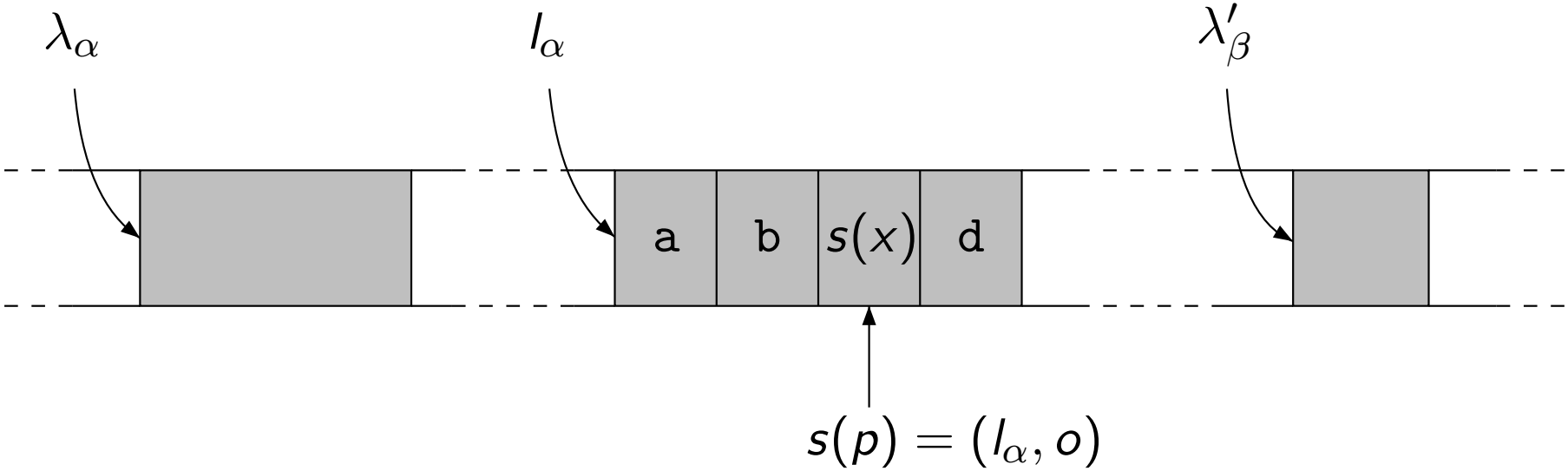
Sémantique des instructions (2)

- pour $*p = x$: si $s(p) \in \mathrm{dom}(h)$, *avant*



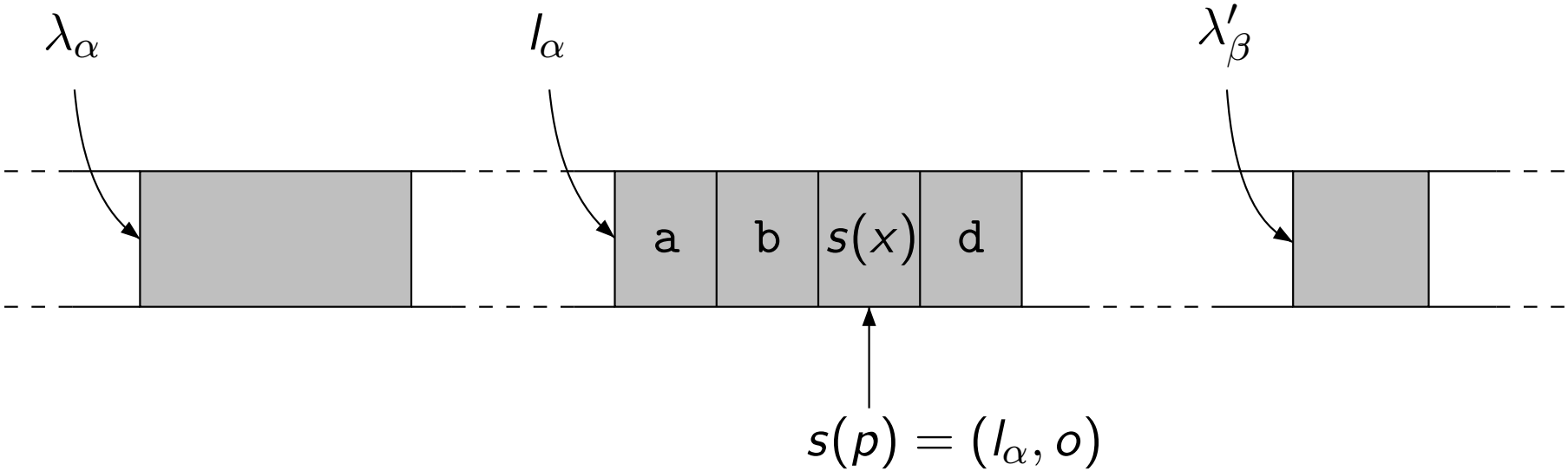
Sémantique des instructions (2)

- pour $*p = x$: si $s(p) \in \mathrm{dom}(h)$, *après*



Sémantique des instructions (2)

- pour $*p = x$: si $s(p) \in \mathrm{dom}(h)$, *après*



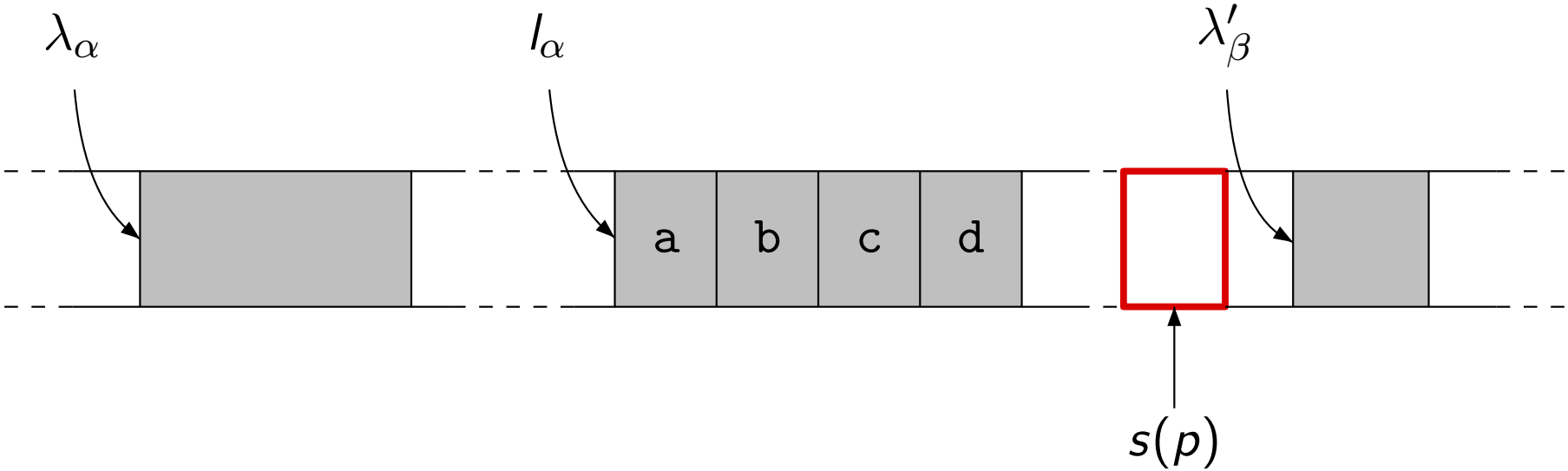
Ecriture valide, transition vers :

$$s' = s$$
$$h' = h[(l_\alpha, o) \mapsto s(x)]$$

où $s(p) = (l_\alpha, o)$.

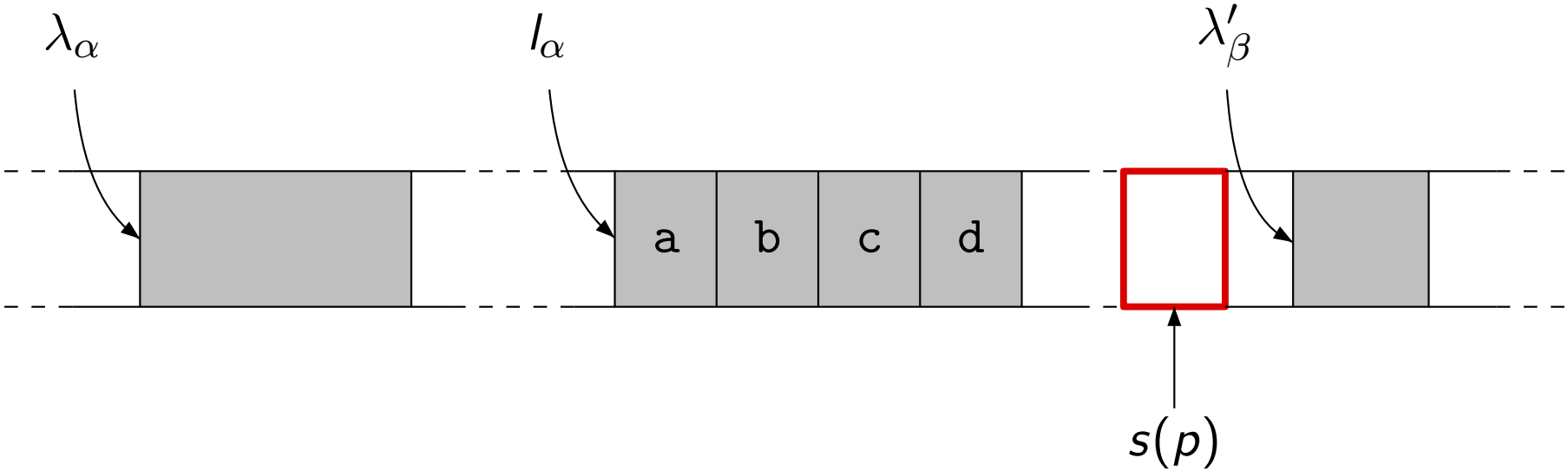
Sémantique des instructions (2)

- pour $*p = x$: si $s(p) \notin \mathrm{dom}(h)$, *avant*



Sémantique des instructions (2)

- pour $*p = x$: si $s(p) \notin \mathrm{dom}(h)$, *avant*



Si $s(p) = (l_\alpha, o) \notin \mathrm{dom}(h)$ ou $s(p) = \omega$, alors il y a un *heap overflow* :

pas de transition possible vers un autre état

Plan de l'exposé

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i) \text{ et } s(n) \text{ quelconques}\}$
$2:$	$i = 0;$	$2 \mapsto \emptyset$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \emptyset$
$4:$	$\quad i = i + 1;$	$4 \mapsto \emptyset$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \emptyset$
$4:$	$\quad i = i + 1;$	$4 \mapsto \emptyset$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) = 0$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \emptyset$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) = 0$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) = 0$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$$
\begin{array}{ll}
1: & n = 10; \\
2: & i = 0; \\
3: & \text{while } (i < n)\ \{ \\
4: & \quad i = i + 1; \\
5: & \}
\end{array}
\qquad
\begin{array}{l}
1 \mapsto \{s(i) \text{ et } s(n) \text{ quelconques}\} \\
2 \mapsto \{s(n) = 10 \text{ et } s(i) \text{ quelconque}\} \\
3 \mapsto \{s(i) = 0 \text{ et } s(n) = 10\} \\
\qquad\quad \cup \{s(i) = 1 \text{ et } s(n) = 10\} \\
4 \mapsto \{s(i) = 0 \text{ et } s(n) = 10\} \\
5 \mapsto \emptyset
\end{array}
$$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) \in [0; 1]$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) = 0$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) \in [0; 1]$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) \in [0; 1]$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:\quad n = 10;$	$1 \mapsto \{s(i) \text{ et } s(n) \text{ quelconques}\}$
$2:\quad i = 0;$	$2 \mapsto \{s(n) = 10 \text{ et } s(i) \text{ quelconque}\}$
$3:\quad \text{while } (i < n)\ \{$	$3 \mapsto \{s(i) = 0 \text{ et } s(n) = 10\}$
$4:\qquad i = i + 1;$	$\{s(i) \in [1;2] \text{ et } s(n) = 10\}$
$5:\quad \}$	$4 \mapsto \{s(i) \in [0;1] \text{ et } s(n) = 10\}$
	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) \in [0; 2]$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) \in [0; 1]$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) \in [0; 2]$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) \in [0; 2]$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) \in [0; 3]$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) \in [0; 2]$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) \in [0; 3]$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) \in [0; 3]$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) \in [0; 4]$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) \in [0; 3]$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) \in [0; 4]$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) \in [0; 4]$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) \in [0; 9]$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) \in [0; 9]$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) \in [0; 10]$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) \in [0; 9]$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) \in [0; 10]$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) \in [0; 9]$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \{s(i) = 10$ et $s(n) = 10\}$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) \in [0; 10]$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) \in [0; 9]$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \{s(i) = 10$ et $s(n) = 10\}$

Un point fixe est atteint :
la sémantique collectrice $\mathcal{C}(P)$

Sémantique collectrice

Collecter les états de la mémoire atteignables à chaque point de contrôle du programme P.

$1:$	$n = 10;$	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0;$	$2 \mapsto \{s(n) = 10$ et $s(i)$ quelconque$\}$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \{s(i) \in [0; 10]$ et $s(n) = 10\}$
$4:$	$\quad i = i + 1;$	$4 \mapsto \{s(i) \in [0; 9]$ et $s(n) = 10\}$
$5:$	$\}$	$5 \mapsto \{s(i) = 10$ et $s(n) = 10\}$

Absence de *heap overflow*

$$\vdots$$
$$c: \quad *p = e;$$
$$\vdots$$

Pas de *heap overflow* ssi pour tout $s \dagger h$ dans $\mathcal{C}(P)(c)$,

$$s(p) = (l_\alpha, o) \in \mathrm{dom}(h)$$

Sémantique collectrice (3)

Problème : la sémantique collectrice d'un programme P n'est pas calculable en général.

1 :	assert($n \geq 0$);	$1 \mapsto \emptyset$
2 :	$i = 0$;	$2 \mapsto \emptyset$
3 :	while $(i < n)$ {	$3 \mapsto \emptyset$
4 :	$\quad i = i + 1$;	$4 \mapsto \emptyset$
5 :	}	$5 \mapsto \emptyset$

Sémantique collectrice (3)

Problème : la sémantique collectrice d'un programme P n'est pas calculable en général.

$1:$	assert$(n \geq 0)$;	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
$2:$	$i = 0$;	$2 \mapsto \emptyset$
$3:$	while $(i < n)$ $\{$	$3 \mapsto \emptyset$
$4:$	$\quad i = i + 1$;	$4 \mapsto \emptyset$
$5:$	$\}$	$5 \mapsto \emptyset$

Sémantique collectrice (3)

Problème : la sémantique collectrice d'un programme P n'est pas calculable en général.

1 :	assert($n \geq 0$);	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
2 :	$i = 0$;	$2 \mapsto \{s(n) \geq 0$ et $s(i)$ quelconque$\}$
3 :	while $(i < n)$ {	$3 \mapsto \emptyset$
4 :	$\quad i = i + 1$;	$4 \mapsto \emptyset$
5 :	}	$5 \mapsto \emptyset$

Sémantique collectrice (3)

Problème : la sémantique collectrice d'un programme P n'est pas calculable en général.

1 :	assert$(n \geq 0)$;	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
2 :	$i = 0$;	$2 \mapsto \{s(n) \geq 0$ et $s(i)$ quelconque$\}$
3 :	while $(i < n)$ {	$3 \mapsto \{s(i) = 0$ et $s(n) \geq 0\}$
4 :	$\quad i = i + 1$;	$4 \mapsto \emptyset$
5 :	}	$5 \mapsto \emptyset$

Sémantique collectrice (3)

Problème : la sémantique collectrice d'un programme P n'est pas calculable en général.

1 :	assert$(n \geq 0)$;	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
2 :	$i = 0$;	$2 \mapsto \{s(n) \geq 0$ et $s(i)$ quelconque$\}$
3 :	while $(i < n)$ {	$3 \mapsto \{s(i) = 0$ et $s(n) \geq 0\}$
4 :	$\quad i = i + 1$;	$4 \mapsto \{s(i) = 0$ et $s(n) \geq 1\}$
5 :	}	$5 \mapsto \{s(i) = s(n) = 0\}$

Sémantique collectrice (3)

Problème : la sémantique collectrice d'un programme P n'est pas calculable en général.

1 :	assert($n \geq 0$);	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
2 :	$i = 0$;	$2 \mapsto \{s(n) \geq 0$ et $s(i)$ quelconque$\}$
3 :	while $(i < n)$ {	$3 \mapsto \{s(i) \in [0; 1]$ et $s(i) \leq s(n)\}$
4 :	$\quad i = i + 1$;	$4 \mapsto \{s(i) = 0$ et $s(n) \geq 1\}$
5 :	}	$5 \mapsto \{s(i) = s(n) = 0\}$

Sémantique collectrice (3)

Problème : la sémantique collectrice d'un programme P n'est pas calculable en général.

1 :	assert$(n \geq 0)$;	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
2 :	$i = 0$;	$2 \mapsto \{s(n) \geq 0$ et $s(i)$ quelconque$\}$
3 :	while $(i < n)$ {	$3 \mapsto \{s(i) \in [0;1]$ et $s(i) \leq s(n)\}$
4 :	$\quad i = i + 1$;	$4 \mapsto \{s(i) \in [0;1]$ et $s(i) < s(n)\}$
5 :	}	$5 \mapsto \{s(i) = s(n) \in [0;1]\}$

Sémantique collectrice (3)

Problème : la sémantique collectrice d'un programme P n'est pas calculable en général.

1 :	assert($n \geq 0$);	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
2 :	$i = 0$;	$2 \mapsto \{s(n) \geq 0$ et $s(i)$ quelconque$\}$
3 :	while $(i < n)$ {	$3 \mapsto \{s(i) \in [0; 2]$ et $s(i) \leq s(n)\}$
4 :	$\quad i = i + 1$;	$4 \mapsto \{s(i) \in [0; 1]$ et $s(i) < s(n)\}$
5 :	}	$5 \mapsto \{s(i) = s(n) \in [0; 1]\}$

Sémantique collectrice (3)

Problème : la sémantique collectrice d'un programme P n'est pas calculable en général.

1 :	assert($n \geq 0$);	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
2 :	$i = 0$;	$2 \mapsto \{s(n) \geq 0$ et $s(i)$ quelconque$\}$
3 :	while $(i < n)$ {	$3 \mapsto \{s(i) \in [0; 2]$ et $s(i) \leq s(n)\}$
4 :	$\quad i = i + 1$;	$4 \mapsto \{s(i) \in [0; 2]$ et $s(i) < s(n)\}$
5 :	}	$5 \mapsto \{s(i) = s(n) \in [0; 2]\}$

Sémantique collectrice (3)

Problème : la sémantique collectrice d'un programme P n'est pas calculable en général.

1 :	assert($n \geq 0$);	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
2 :	$i = 0$;	$2 \mapsto \{s(n) \geq 0$ et $s(i)$ quelconque$\}$
3 :	while $(i < n)$ {	$3 \mapsto \{s(i) \in [0; 3]$ et $s(i) \leq s(n)\}$
4 :	$\quad i = i + 1$;	$4 \mapsto \{s(i) \in [0; 2]$ et $s(i) < s(n)\}$
5 :	}	$5 \mapsto \{s(i) = s(n) \in [0; 2]\}$

Sémantique collectrice (3)

Problème : la sémantique collectrice d'un programme P n'est pas calculable en général.

1 :	assert$(n \geq 0)$;	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
2 :	$i = 0$;	$2 \mapsto \{s(n) \geq 0$ et $s(i)$ quelconque$\}$
3 :	while $(i < n)$ {	$3 \mapsto \{s(i) \in [0; 3]$ et $s(i) \leq s(n)\}$
4 :	$\quad i = i + 1$;	$4 \mapsto \{s(i) \in [0; 3]$ et $s(i) < s(n)\}$
5 :	}	$5 \mapsto \{s(i) = s(n) \in [0; 3]\}$

Sémantique collectrice (3)

Problème : la sémantique collectrice d'un programme P n'est pas calculable en général.

1 :	assert($n \geq 0$);	$1 \mapsto \{s(i)$ et $s(n)$ quelconques$\}$
2 :	$i = 0$;	$2 \mapsto \{s(n) \geq 0$ et $s(i)$ quelconque$\}$
3 :	while $(i < n)$ {	$3 \mapsto \{0 \leq s(i) \leq s(n)\}$
4 :	$i = i + 1$;	$4 \mapsto \{0 \leq s(i) < s(n)\}$
5 :	}	$5 \mapsto \{s(i) = s(n) \geq 0\}$

$\underbrace{\qquad\qquad}_{=\mathcal{C}(P)}$

- états non représentables en machine.
- transitions non calculables par un algorithme.
- point fixe $\mathcal{C}(P)$ atteint en un nombre infini d'étapes.

Vers l'analyse

Grâce à des approximations, chacun des problèmes peut être résolu.

Ces approximations ne doivent « oublier » aucun état accessible lors d'une exécution $\longrightarrow$ *sur-approximations*.
C'est sur ce principe que repose l'interprétation abstraite [2].

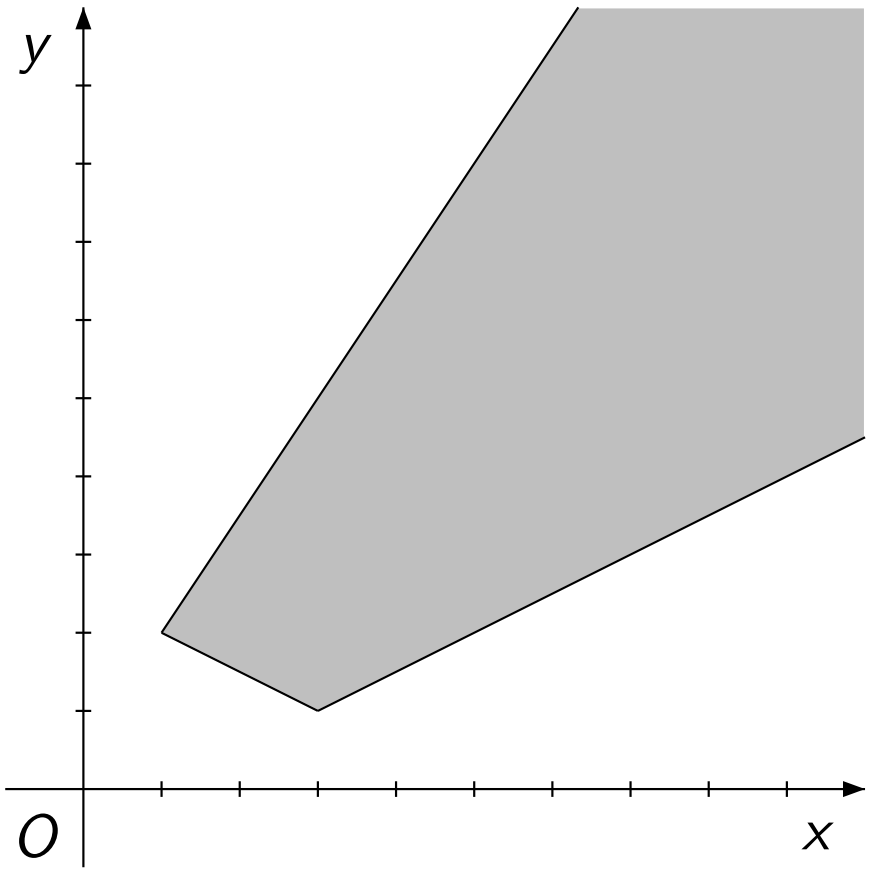
L'algorithme approché retournera :

- « il est certain que le programme ne provoque pas d'erreurs ».
- ou « je ne sais pas ».

Plan de l'exposé

Plan de l'exposé

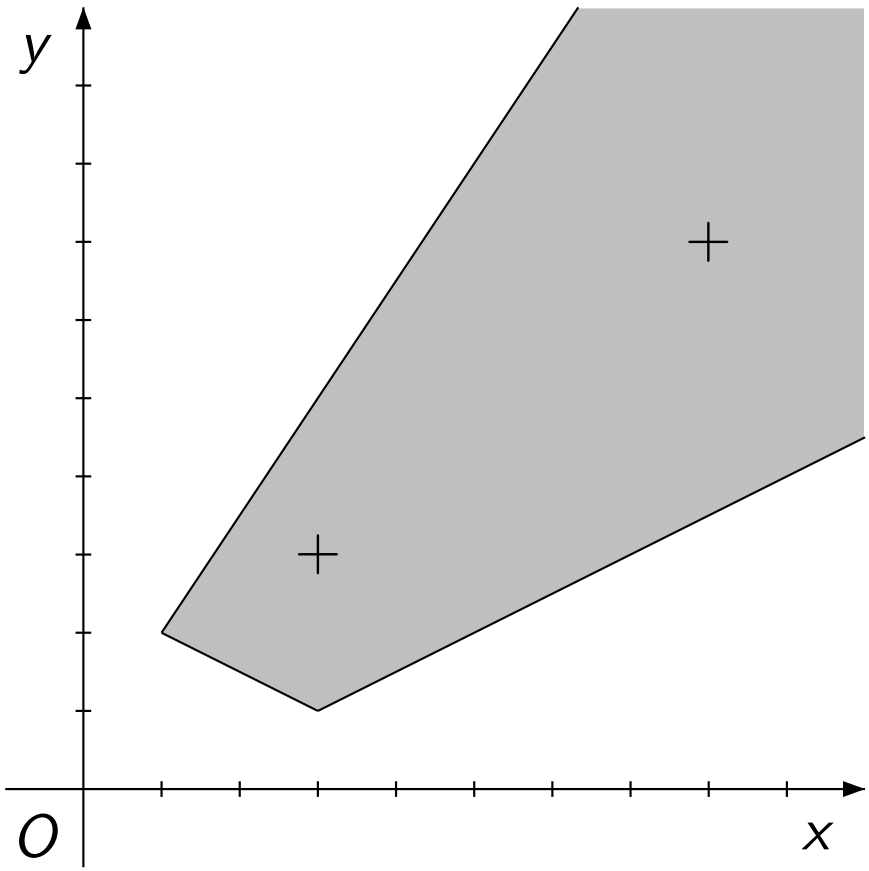
Exemple : approximations par des polyèdres convexes [3]



Représentable en machine par :

$$\begin{cases} 2y - 3x \leq 1 \\ 2y + x \geq 5 \\ 2y - x \geq -1 \end{cases}$$

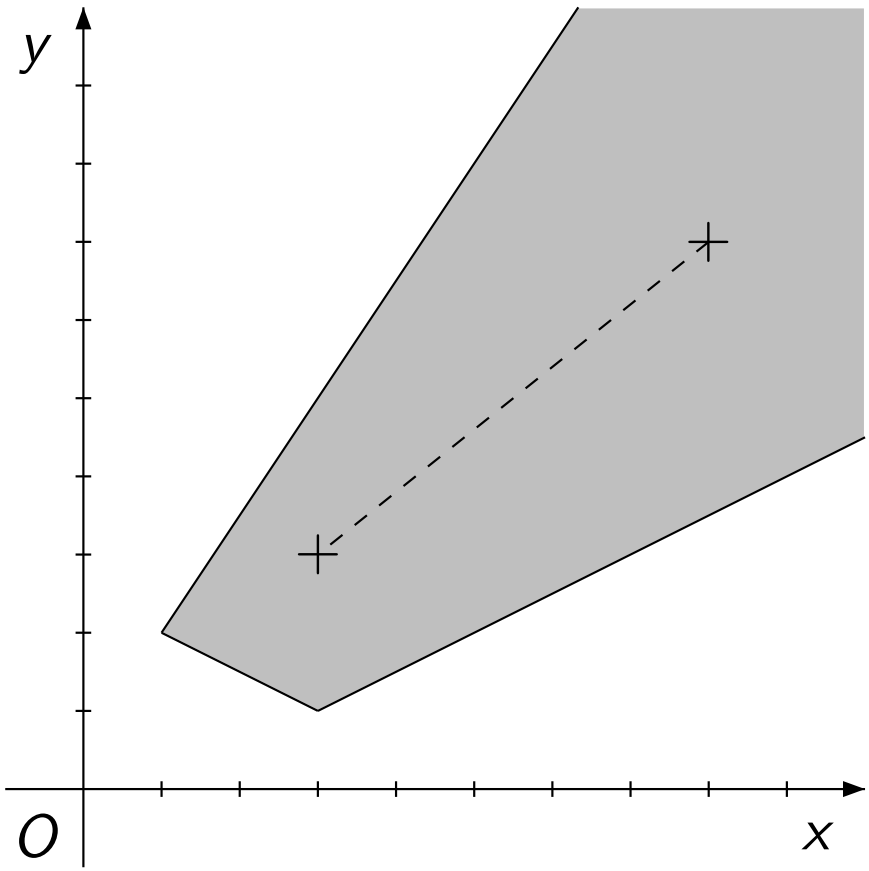
Exemple : approximations par des polyèdres convexes [3]



Représentable en machine par :

$$\begin{cases} 2y - 3x \leq 1 \\ 2y + x \geq 5 \\ 2y - x \geq -1 \end{cases}$$

Exemple : approximations par des polyèdres convexes [3]

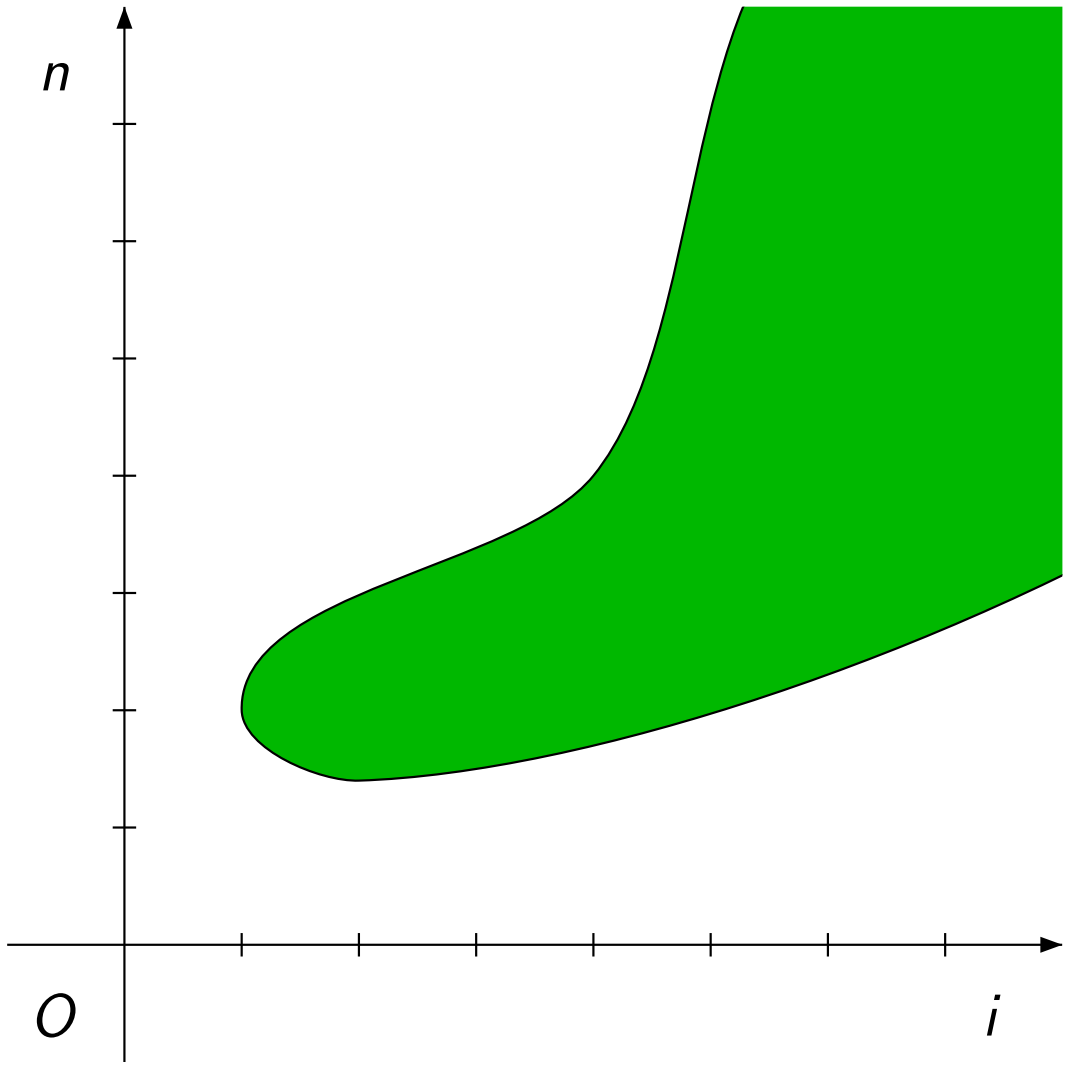


Représentable en machine par :

$$\begin{cases} 2y - 3x \leq 1 \\ 2y + x \geq 5 \\ 2y - x \geq -1 \end{cases}$$

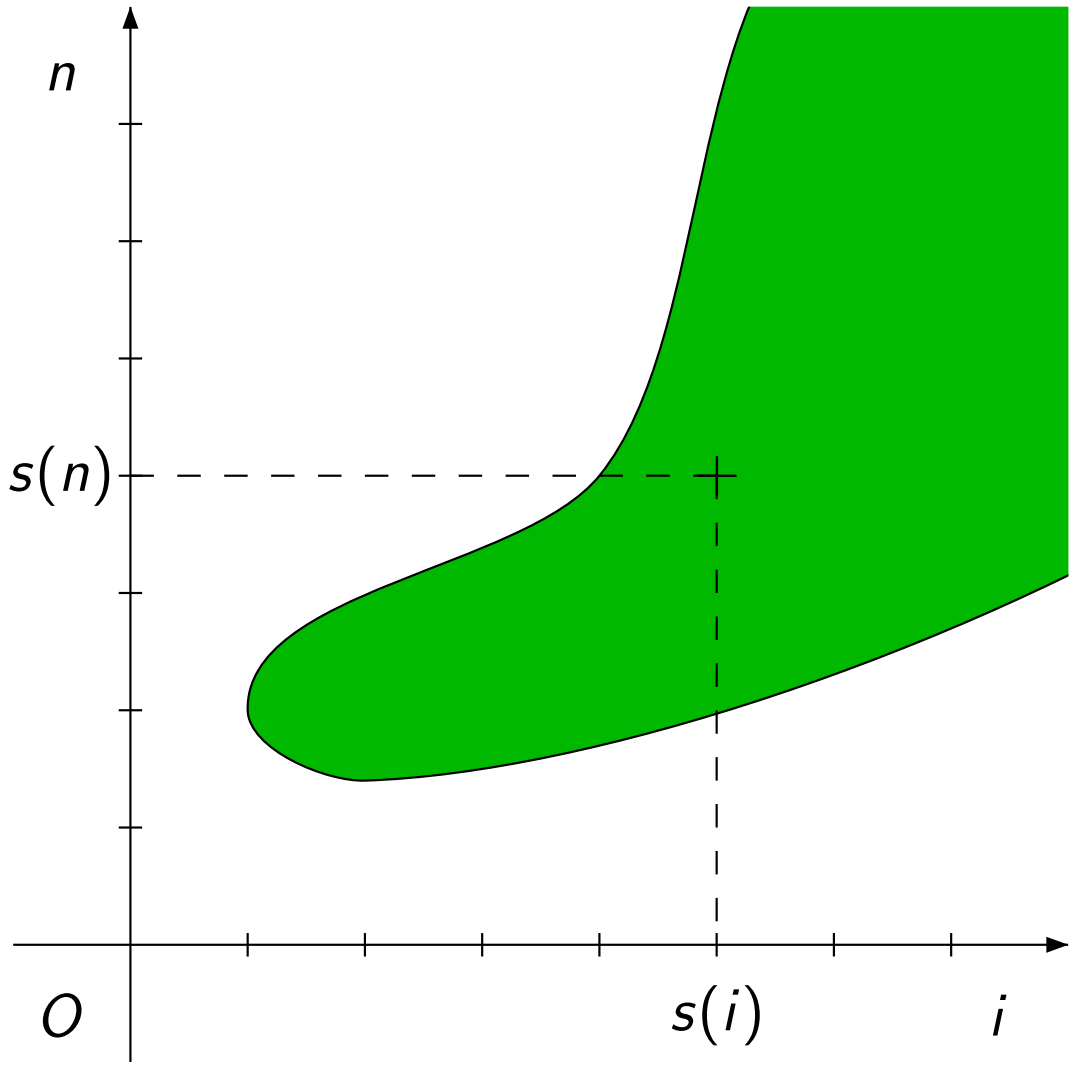
Exemple : approximations par des polyèdres convexes (2)

Programme avec les variables i et n de type **uchar**.



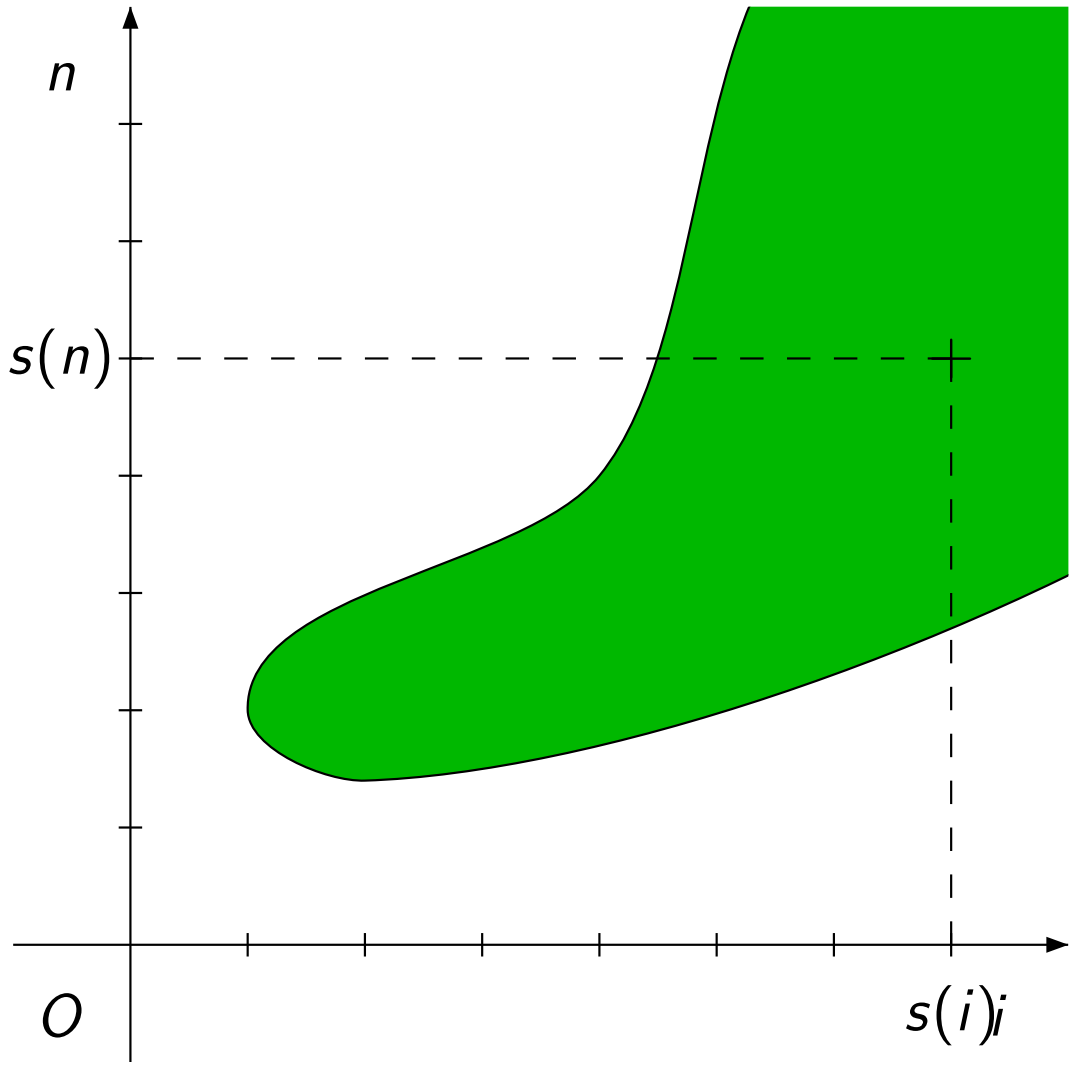
Exemple : approximations par des polyèdres convexes (2)

Programme avec les variables i et n de type **uchar**.



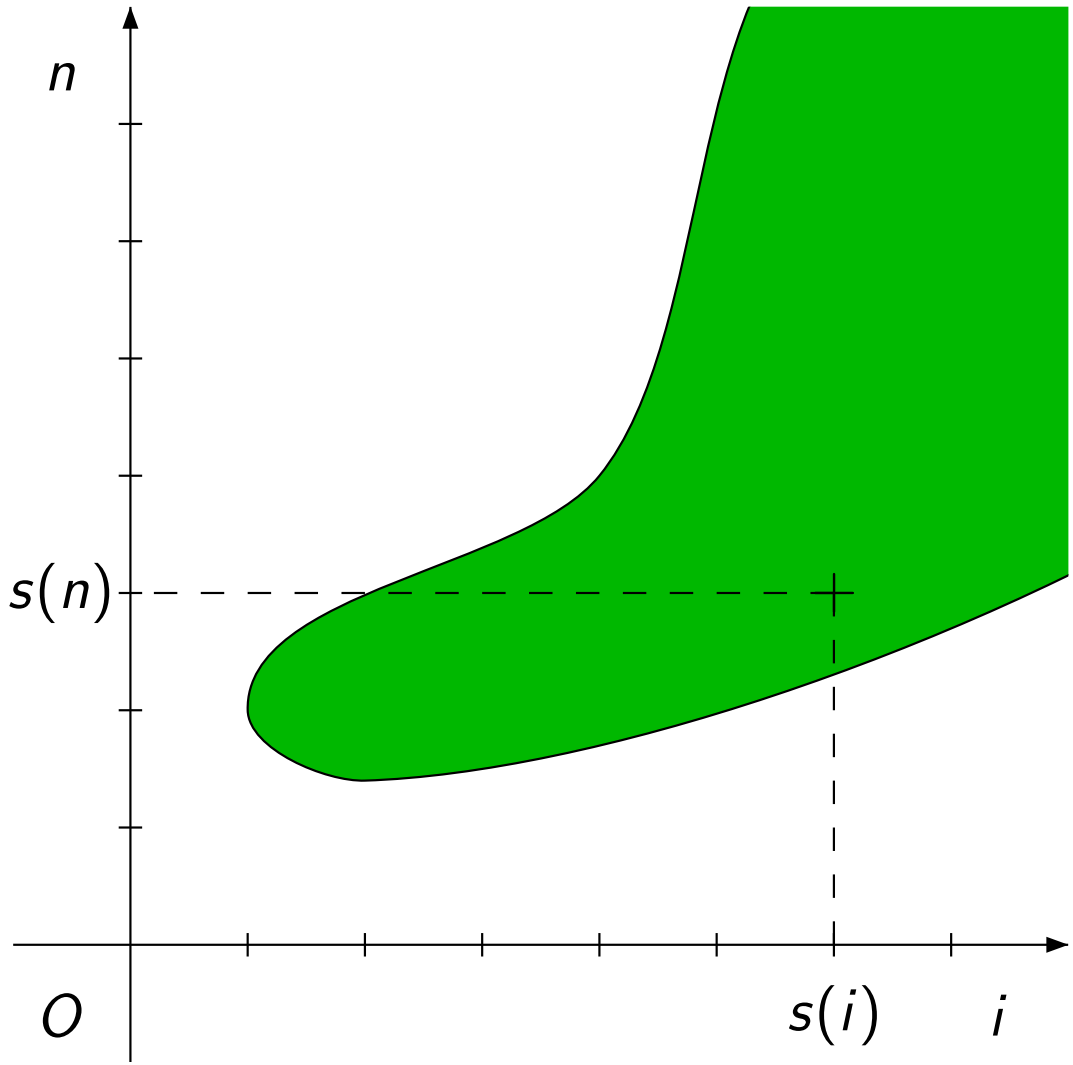
Exemple : approximations par des polyèdres convexes (2)

Programme avec les variables i et n de type **uchar**.



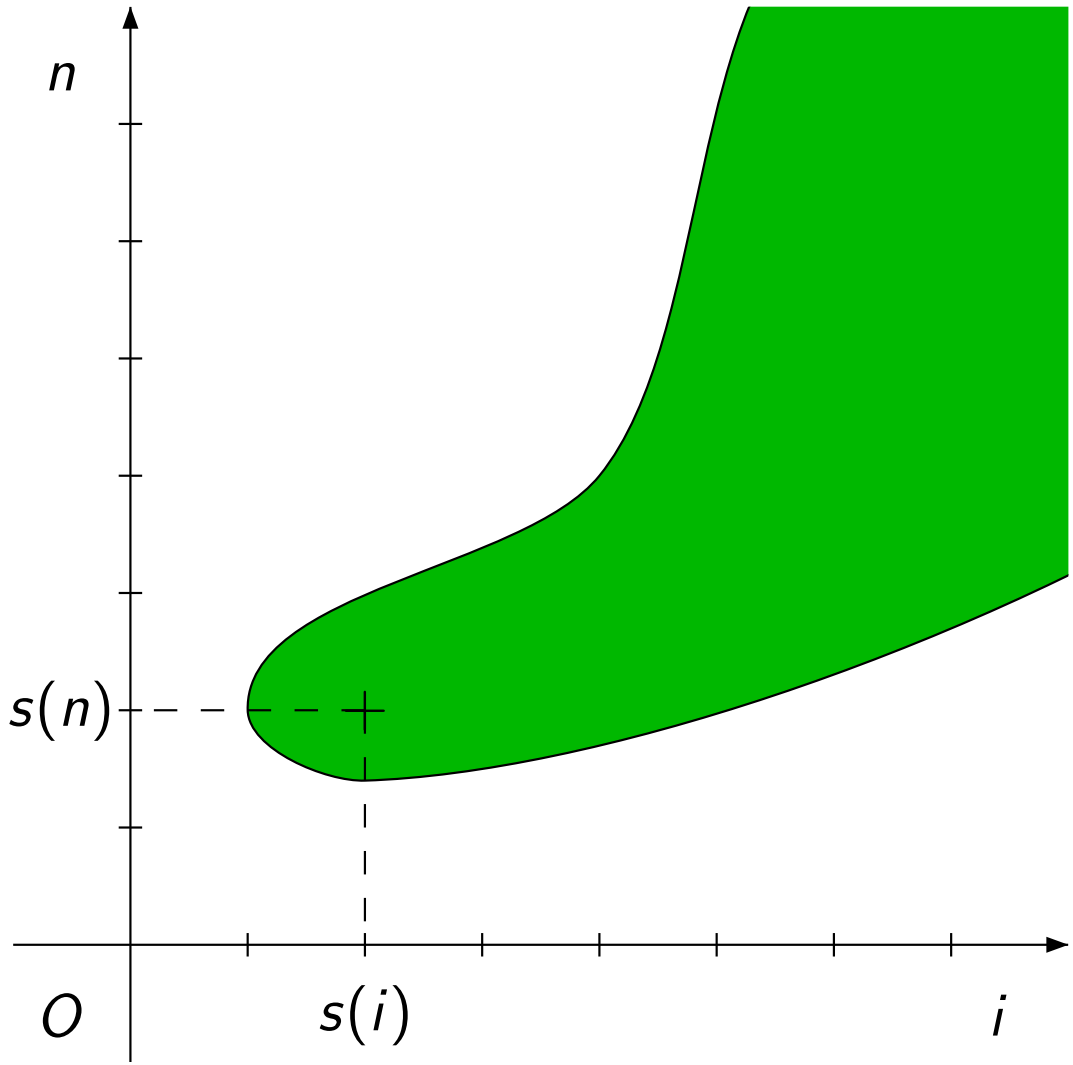
Exemple : approximations par des polyèdres convexes (2)

Programme avec les variables i et n de type **uchar**.



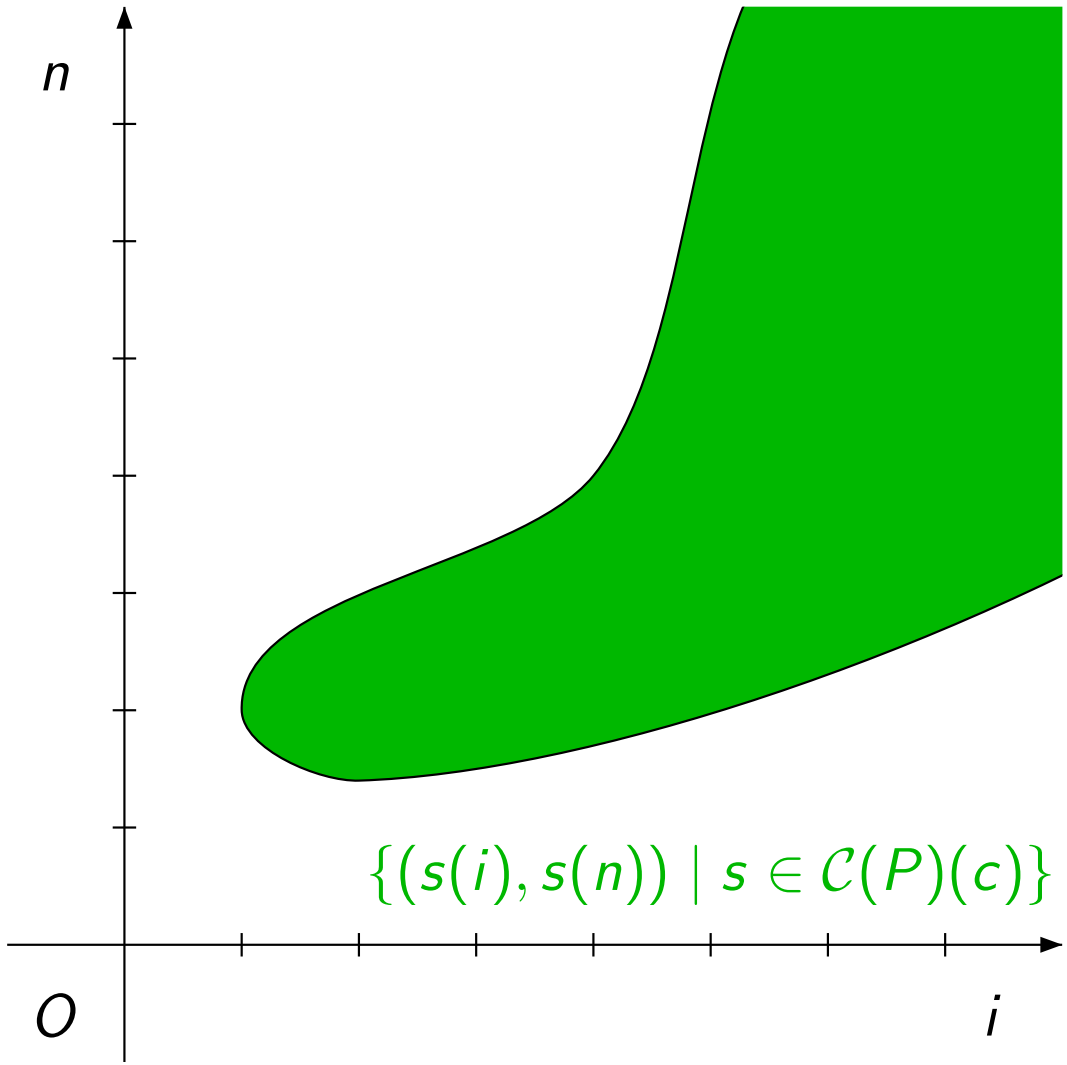
Exemple : approximations par des polyèdres convexes (2)

Programme avec les variables i et n de type **uchar**.



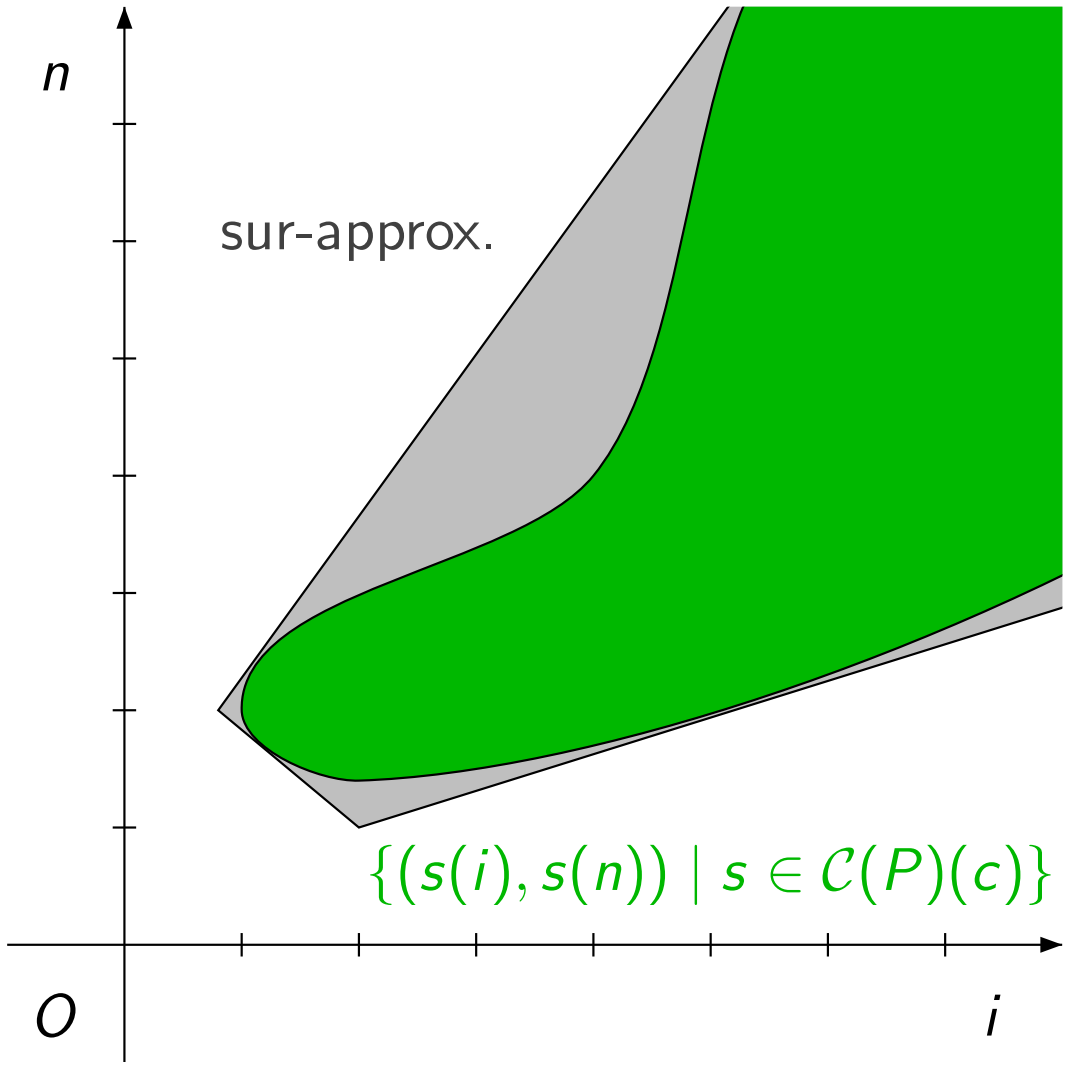
Exemple : approximations par des polyèdres convexes (2)

Programme avec les variables i et n de type **uchar**.



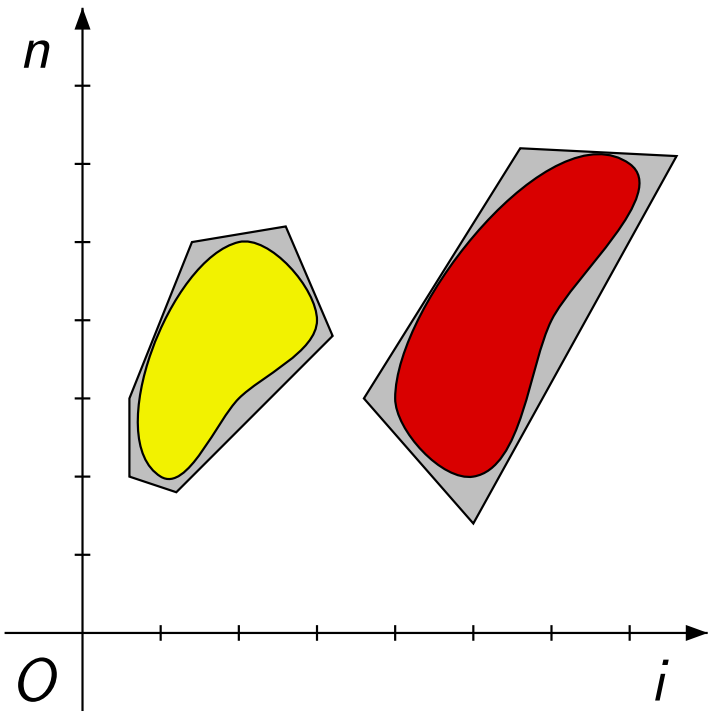
Exemple : approximations par des polyèdres convexes (2)

Programme avec les variables i et n de type **uchar**.



Exemple : approximations par des polyèdres convexes (3)

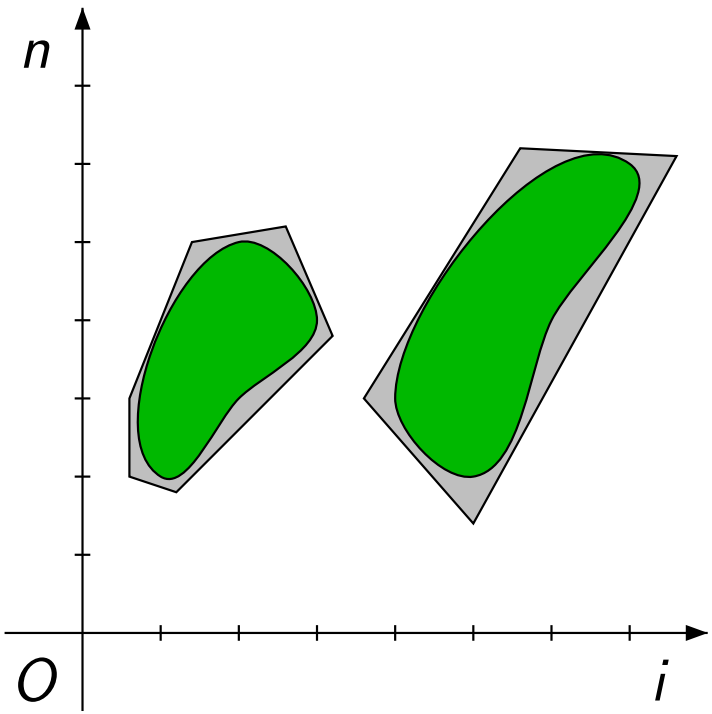
Pour chaque opération dans le « concret », une opération dans l'« abstrait ».




```
1 :     if ... {
            ⋮
3 :     }
4 :     else {
            ⋮
9 :     }
10 :
```

Exemple : approximations par des polyèdres convexes (3)

Pour chaque opération dans le « concret », une opération dans l'« abstrait ».

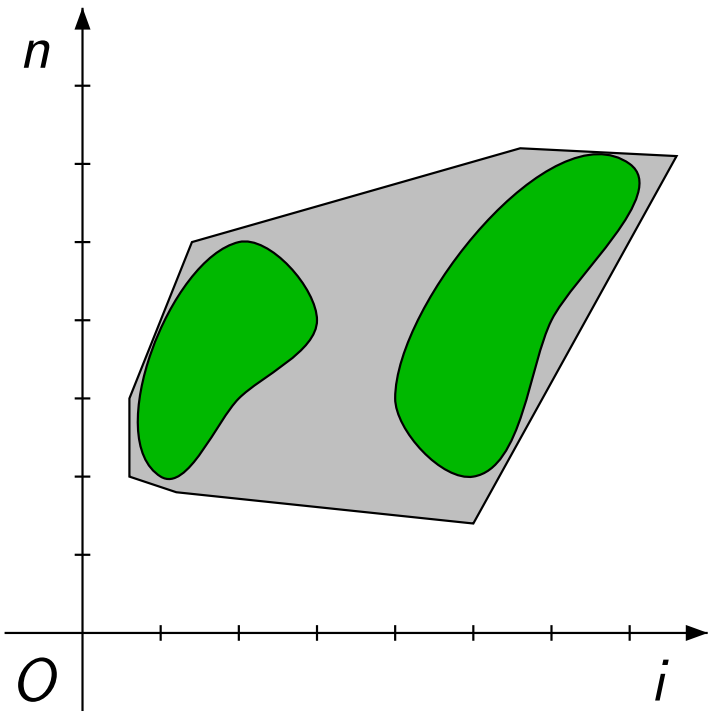



```
1 :     if ... {
            ⋮
3 :     }
4 :     else {
            ⋮
9 :     }
10 :
```

Au point de contrôle 10, états au point 3 + états au point 9.

Exemple : approximations par des polyèdres convexes (3)

Pour chaque opération dans le « concret », une opération dans l'« abstrait ».




```
1 :     if ... {
            ⋮
3 :     }
4 :     else {
            ⋮
9 :     }
10 :
```

Au point de contrôle 10, états au point 3 + états au point 9.

Plan de l'exposé

Un peu de formalisme

Une abstraction consiste en un domaine abstrait $\mathcal{D}$, muni d'un opérateur de concrétisation γ : pour tout $\mathcal{X}$ dans $\mathcal{D}$

$$\mathcal{X} \text{ sur-approxime } \gamma(\mathcal{X})$$

Exemple

Pour les polyèdres sur les variables i et n,

$$\gamma(\mathcal{P}) = \{s \mid (s(i), s(n)) \in \mathcal{P}\}$$

Pour chaque opération F dans le « concret », on a besoin d'une opération correspondante $\mathcal{F}$ dans l'« abstrait », qui conserve la sur-approximation :

$$F(\gamma(\mathcal{X})) \subseteq \gamma(\mathcal{F}(\mathcal{X}))$$

Un peu de formalisme

Une abstraction consiste en un domaine abstrait $\mathcal{D}$, muni d'un opérateur de concrétisation γ : pour tout $\mathcal{X}$ dans $\mathcal{D}$

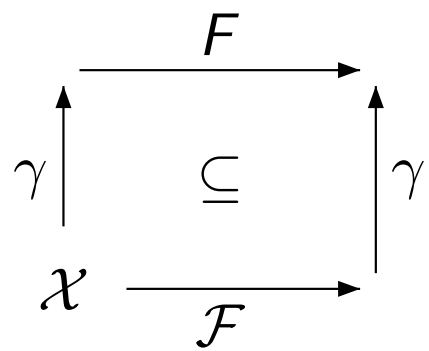
$$\mathcal{X} \text{ sur-approxime } \gamma(\mathcal{X})$$

Exemple

Pour les polyèdres sur les variables i et n,

$$\gamma(\mathcal{P}) = \{s \mid (s(i), s(n)) \in \mathcal{P}\}$$

Pour chaque opération F dans le « concret », on a besoin d'une opération correspondante $\mathcal{F}$ dans l'« abstrait », qui conserve la sur-approximation :



Un peu de formalisme

Une abstraction consiste en un domaine abstrait $\mathcal{D}$, muni d'un opérateur de concrétisation γ : pour tout $\mathcal{X}$ dans $\mathcal{D}$

$$\mathcal{X} \text{ sur-approxime } \gamma(\mathcal{X})$$

Exemple

Pour les polyèdres sur les variables i et n,

$$\gamma(\mathcal{P}) = \{s \mid (s(i), s(n)) \in \mathcal{P}\}$$

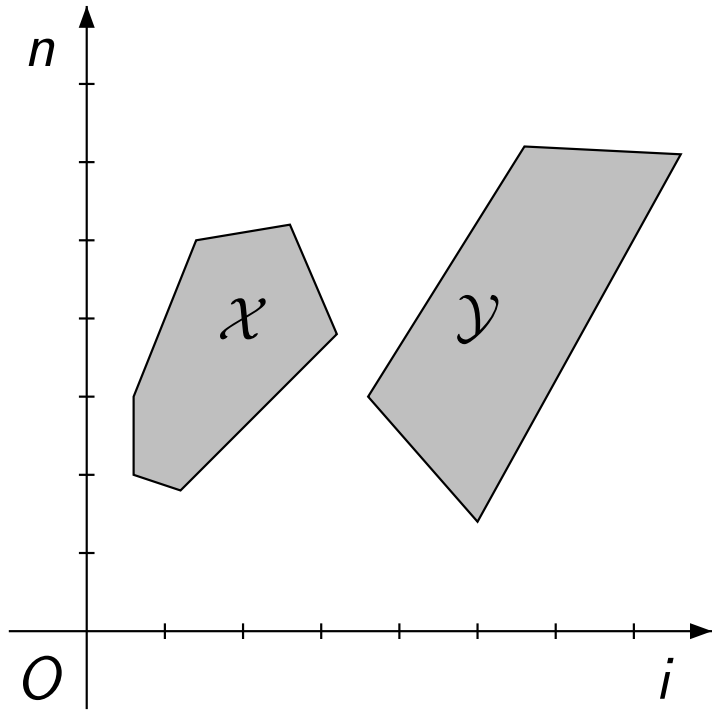
Pour chaque opération F dans le « concret », on a besoin d'une opération correspondante $\mathcal{F}$ dans l'« abstrait », qui conserve la sur-approximation :

Exemple

Union abstraite $\sqcup$: $\gamma(\mathcal{X}) \cup \gamma(\mathcal{Y}) \subseteq \gamma(\mathcal{X} \sqcup \mathcal{Y})$

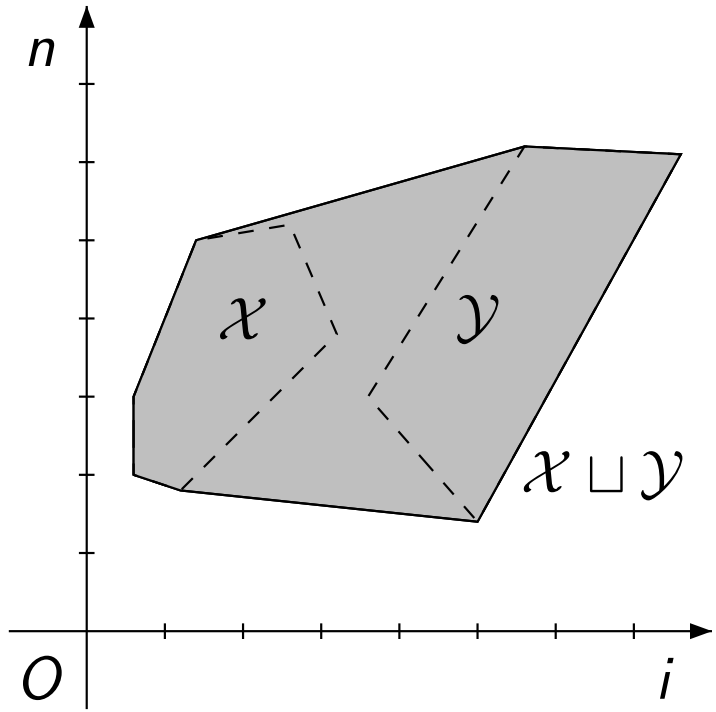
Un peu de formalisme (2)

Dans le cas des polyèdres,



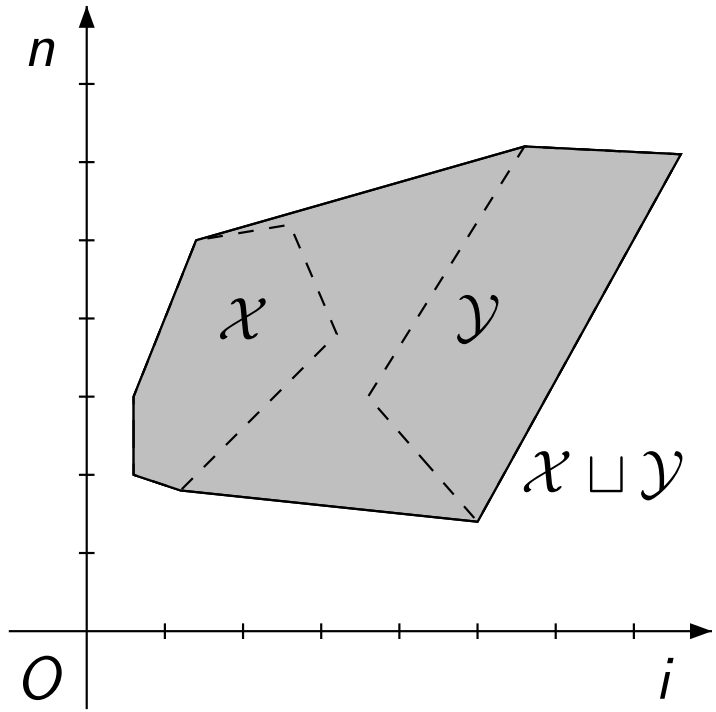
Un peu de formalisme (2)

Dans le cas des polyèdres,



Un peu de formalisme (2)

Dans le cas des polyèdres,



On dispose également :

- d'une opération abstraite d'affectation. Par exemple,

$$\{s' \mid s \in \gamma(\mathcal{X}) \text{ et } s'(n) = s(i) + 1\} \subseteq \gamma(\llparenthesis n \leftarrow i + 1 \rrparenthesis(\mathcal{X}))$$

- d'une opération abstraite de test d'une condition. Par exemple,

$$\{s \mid s \in \gamma(\mathcal{X}) \text{ et } s(i) \leq s(n)\} \subseteq \gamma(\llparenthesis i \leq n \rrparenthesis(\mathcal{X}))$$

Plan de l'exposé

Principe de l'abstraction

- oublier le contenu des *buffer* du tas, ne garder que leur taille.
- fusionner tous les blocs alloués par le même malloc_α.
- utiliser les polyèdres convexes pour sur-approximer
  - les caractères,
  - les *offset* des pointeurs,
  - les tailles des *buffer*.

Principe de l'abstraction

- oublier le contenu des *buffer* du tas, ne garder que leur taille.
- fusionner tous les blocs alloués par le même malloc_α.
- utiliser les polyèdres convexes pour sur-approximer
  - les caractères,
  - les *offset* des pointeurs,
  - les tailles des *buffer*.

Variables auxiliaires

Pour chaque pointeur p,

- p_ℓ : location de l'adresse pointée par p.
- p_o : l'*offset*.

Pour chaque site d'allocation α, la variable α_s : taille des blocs alloués en α.

Etat abstrait de la mémoire

Un état abstrait de la mémoire est de la forme $(\mathcal{L}, \mathcal{N})$:

- $\mathcal{L}$ associe chaque p_ℓ à un sous-ensemble de $\text{Alloc} \cup \{\omega\}$.

  Exemple : si $\mathcal{L}(p_\ell) = \{\beta, \omega\}$,
  - soit $s(p) = \omega$.
  - soit $s(p) = (\lambda_\beta, \cdot)$.

- $\mathcal{N}$ est un polyèdre convexe sur les variables de type **uchar**, les p_o et les α_s.

  Exemple : si $\mathcal{N} = \{0 \leq i < n, p_o = i, \beta_s = n\}$, cela signifie
  - $0 \leq s(i) < s(n)$.
  - $s(p) = (\cdot, s(i))$.
  - et tout bloc dans h de location λ_β a une taille égale à $s(n)$.

Etat abstrait de la mémoire (2)

Exemple

Avec

$$(\mathcal{L}, \mathcal{N}) = \left(\begin{array}{l} \{p_\ell \mapsto \{\alpha, \beta\}\}, \\ \{n \geq 0, p_o = 0, n \leq \alpha_s \leq n+1, \beta_s = 1\} \end{array} \right)$$

des états possibles du tas sont :

Etat abstrait de la mémoire (2)

Exemple

Avec

$$(\mathcal{L}, \mathcal{N}) = \left(\begin{array}{l} \{p_\ell \mapsto \{\alpha, \beta\}\}, \\ \{n \geq 0, p_o = 0, n \leq \alpha_s \leq n+1, \beta_s = 1\} \end{array} \right)$$

des états possibles du tas sont :

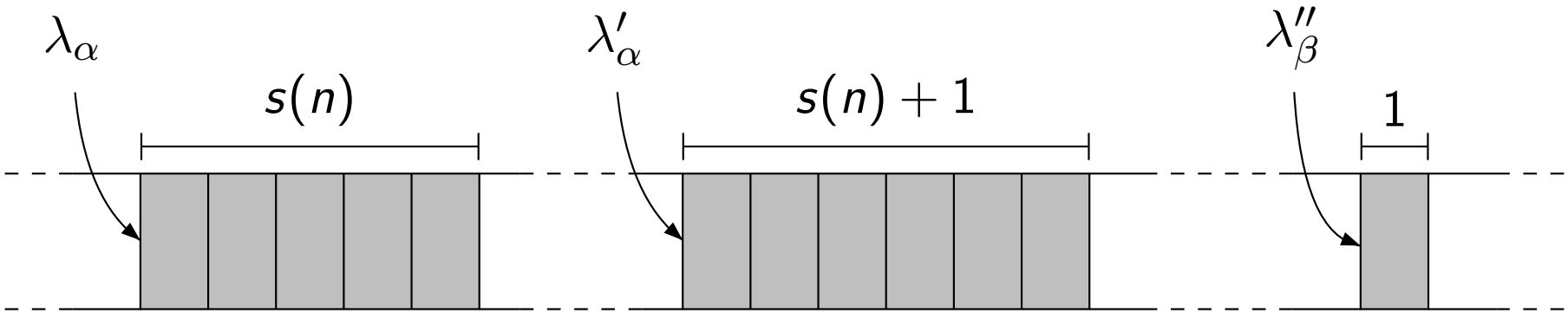
Etat abstrait de la mémoire (2)

Exemple

Avec

$$(\mathcal{L}, \mathcal{N}) = \left(\begin{array}{l} \{p_\ell \mapsto \{\alpha, \beta\}\}, \\ \{n \geq 0, p_o = 0, n \leq \alpha_s \leq n+1, \beta_s = 1\} \end{array} \right)$$

des états possibles du tas sont :



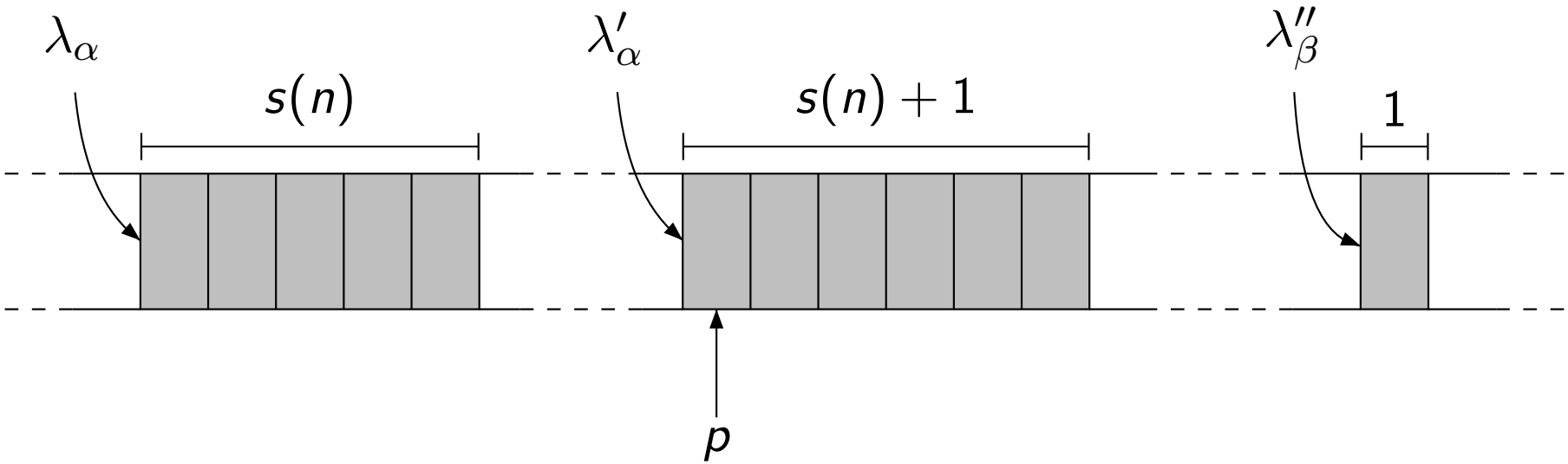
Etat abstrait de la mémoire (2)

Exemple

Avec

$$(\mathcal{L}, \mathcal{N}) = \left(\begin{array}{l} \{p_\ell \mapsto \{\alpha, \beta\}\}, \\ \{n \geq 0, p_o = 0, n \leq \alpha_s \leq n+1, \beta_s = 1\} \end{array} \right)$$

des états possibles du tas sont :



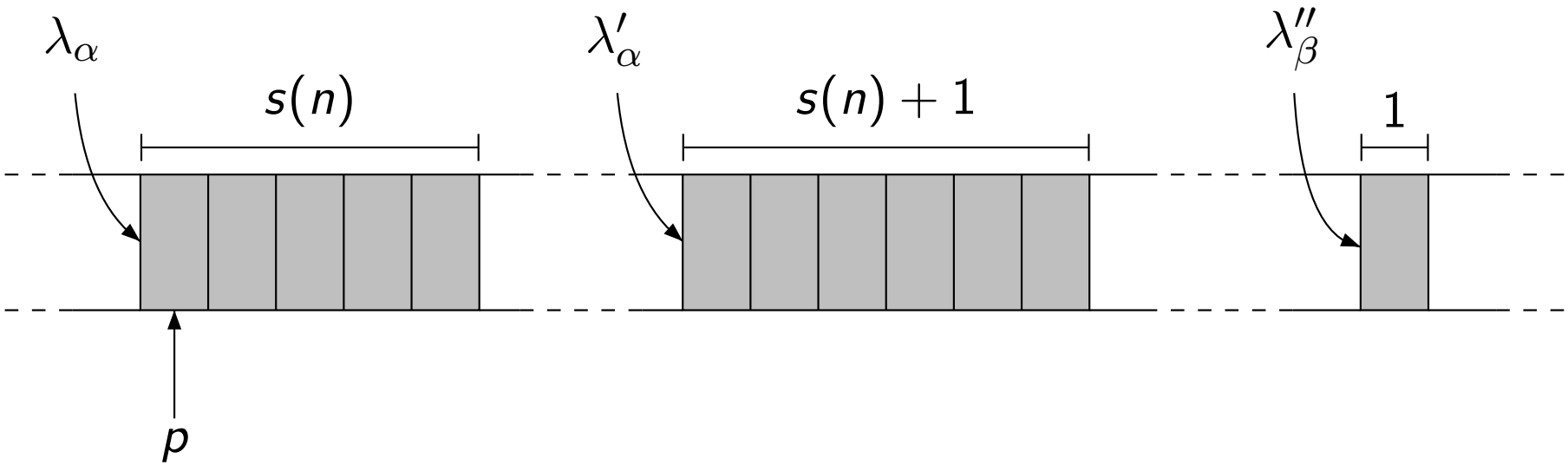
Etat abstrait de la mémoire (2)

Exemple

Avec

$$(\mathcal{L}, \mathcal{N}) = \left(\begin{array}{l} \{p_\ell \mapsto \{\alpha, \beta\}\}, \\ \{n \geq 0, p_o = 0, n \leq \alpha_s \leq n+1, \beta_s = 1\} \end{array} \right)$$

des états possibles du tas sont :

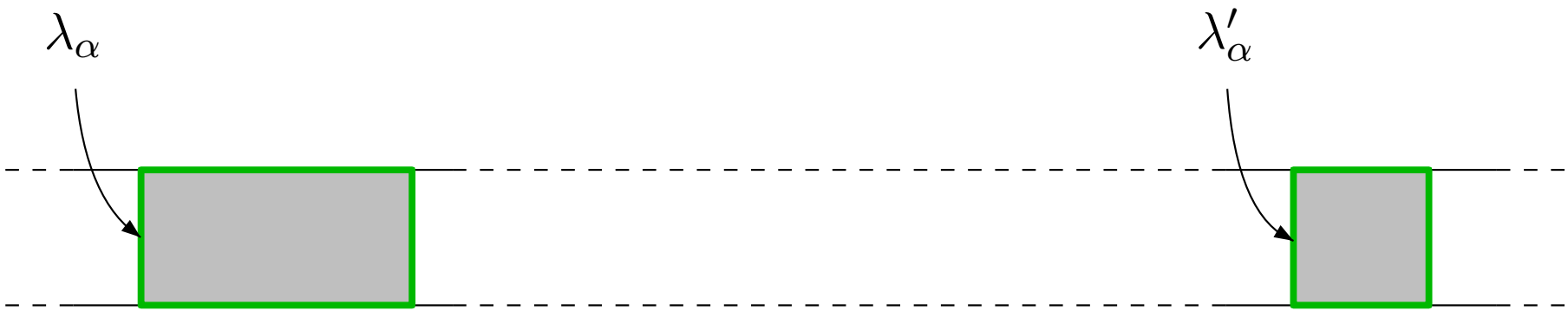


Sémantique abstraite

Quel état abstrait $(\mathcal{L}', \mathcal{N}')$ de la mémoire après l'exécution de *instr* sur l'état abstrait $(\mathcal{L}, \mathcal{N})$?

- pour $p = \mathsf{malloc}_\alpha(n)$:

$$\mathcal{L}' = \mathcal{L}[p_\ell \mapsto \{\alpha\}]$$
$$\mathcal{N}' = (\!|p_o \leftarrow 0|\!)(\underbrace{(\!|\alpha_s \leftarrow n|\!)(\mathcal{N})}_{\text{nouvelle valeur}} \sqcup \underbrace{(\!|\alpha_s \geq 1|\!)(\mathcal{N})}_{\text{anciennes valeurs}})$$

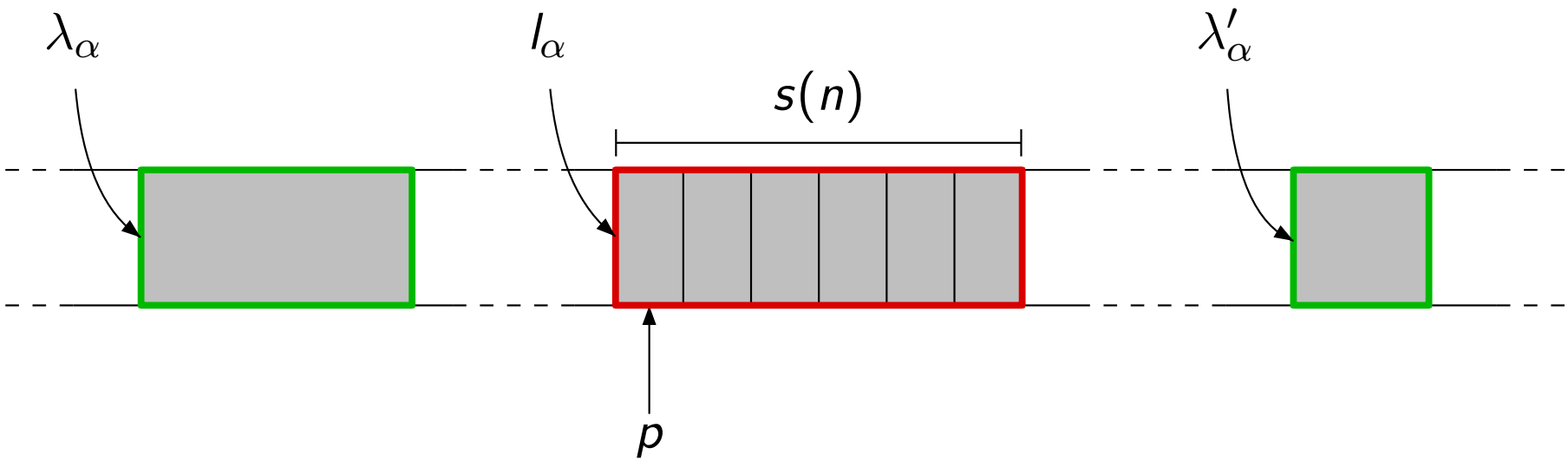


Sémantique abstraite

Quel état abstrait $(\mathcal{L}', \mathcal{N}')$ de la mémoire après l'exécution de *instr* sur l'état abstrait $(\mathcal{L}, \mathcal{N})$?

- pour $p = \mathrm{malloc}_\alpha(n)$:

$$\mathcal{L}' = \mathcal{L}[p_\ell \mapsto \{\alpha\}]$$
$$\mathcal{N}' = (\!|p_o \leftarrow 0|\!)(\underbrace{(\!|\alpha_s \leftarrow n|\!)(\mathcal{N})}_{\text{nouvelle valeur}} \sqcup \underbrace{(\!|\alpha_s \geq 1|\!)(\mathcal{N})}_{\text{anciennes valeurs}})$$



Sémantique abstraite (2)

- pour $*p = x$, l'état reste le même :

$$\mathcal{L}' = \mathcal{L}$$
$$\mathcal{N}' = \mathcal{N}$$

  Suffisant pour montrer que notre exemple ne provoque pas d'erreurs.

Plan de l'exposé

Algorithme de l'analyse

Principe : pour tout point de contrôle, calculer un état abstrait de la mémoire.
En un nombre fini d'étapes, on obtient un point fixe $\mathcal{X}$ vérifiant :

$$\mathcal{C}(P) \subseteq \gamma(\mathcal{X})$$

Absence de *heap overflow*

Pas de *heap overflow* si $\mathcal{X}(c) = (\mathcal{L}, \mathcal{N})$ vérifie

$$\begin{array}{rl} & \vdots \\ c: & *p = e; \\ & \vdots \end{array}$$

$$\underbrace{\omega \notin \mathcal{L}(p_\ell)}_{p \text{ est bien initialisé}}$$

$$\mathcal{N} \subseteq \bigcap_{\alpha_s \in \mathcal{L}(p)} \underbrace{\{0 \leq p_o \leq \alpha_s - 1\}}_{p \text{ est dans les bornes du } \textit{buffer}}$$

Retour sur l'exemple

```
1 :   t = malloc_α(n);
2 :   sz = getuchar();
3 :   if (!(sz − n ≤ 0))
4 :     sz = n;
5 :   i = 0;
6 :   p = t;
7 :   while (i − sz + 1 ≤ 0) {
8 :     tmp = getuchar();
9 :     ∗p = tmp;
10 :    p = p + 1;
11 :    i = i + 1;
12 :  }
```

$$\mathcal{X}(9) = (\{p_\ell \mapsto \{\alpha\}, t_\ell \mapsto \{\alpha\}\}, \{\ldots, 0 \leq i = p_o \leq sz - 1, sz \leq \alpha_s = n\})$$

En particulier,

$$\omega \notin \mathcal{L}_9(p_\ell) \text{ et } 0 \leq p_o \leq \alpha_s - 1$$

$\rightarrow$ pas de *heap overflow*.

Plan de l'exposé

Extensions possibles

- d'autres types entiers, comme les booléens, les entiers signés ou non-signés, les énumérations, divers attributs (comme **short** et **long** en C).
- les pointeurs de profondeur quelconque, les pointeurs vers la pile, les multiples déréférencements.
- les tableaux (multidimensionnels), les structures de données, les unions, les chaînes de caractères [5, 1].
- un flot de contrôle plus élaboré.

Nécessité d'un langage noyau robuste et simple : Newspeak (`http://www.penjili.org/newspeak.html`), et d'un compilateur de C vers Newspeak : C2Newspeak, en LGPL.

Travaux relatifs

- une quantité impressionnante d'outils d'analyse statique qui ne garantissent pas l'absence d'erreurs.
  Exemples : flawfinder, ITS4, Splint, Coverity, BOON, EauClaire, CCA, Uno ...

- pour les analyses statiques sûres,
  - interprétation abstraite
  - *model checking*
  - *predicate abstraction*
  - *theorem proving*

- ce que l'on sait faire en interprétation abstraite :
  - propriétés numériques sur les entiers et les flottants.
  - propriétés numériques sur les pointeurs et les longueurs de chaînes de caractères.
  - propriétés sur la forme de la mémoire.
  - invariants sur les classes dans un langage orienté objets.
  - propriétés de sécurité pour les protocoles cryptographiques.

Conclusion

L'analyse présentée ici :

- permet de montrer l'absence de *heap overflow* dans des programmes réalistes.
- de très nombreuses extensions sont possibles.

Travaux futurs : accroître la précision.

QUESTIONS ?


Xavier Allamigeon, Wenceslas Godard et Charles Hymans :
Static Analysis of String Manipulations in Critical Embedded C Programs.
In Kwangkeun Yi, éditeur : *Static Analysis, 13th International Symposium (SAS'06)*, volume 4134 de *Lecture Notes in Computer Science*, pages 35–51, Seoul, Korea, août 2006. Springer Verlag.


P. Cousot et R. Cousot :
Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints.
In Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.


P. Cousot et N. Halbwachs :

Automatic discovery of linear restraints among variables of a program.
In Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY.


Jack Ganssle :
Big Code.
http://www.embedded.com/showArticle.jhtml?articleID=171


A. Miné :
Field-sensitive value analysis of embedded C programs with union types and pointer arithmetics.
In ACM SIGPLAN LCTES'06, pages 54–63. ACM Press, June 2006.
http://www.di.ens.fr/~mine/publi/article-mine-lctes06.p