Analyse de programmes et complexité implicite

Jean-Yves Moyen

Jean-Yves.Moyen@ens-lyon.org

CNRS – Université Paris 13 (LIPN)

Motivations

- Preuves de terminaison automatiques.
- Caractérisation syntaxiques de classes de complexité.
- Transformation automatique de programmes.

Motivations

- Preuves de terminaison automatiques.
- Caractérisation syntaxiques de classes de complexité.
- Transformation automatique de programmes.

- Caractérisation intensionnelle vs extensionnelle.
- Complexité implicite vs explicite.
- Caractérisation implicite vs explicite.

Complexité d'une fonction

- À chaque programme, sa complexité.
- Pour une fonction, il existe plusieurs programmes la calculant.
- La complexité d'une fonction est le $\min$ des complexités des programmes.

Complexité d’une fonction

- À chaque programme, sa complexité.
- Pour une fonction, il existe plusieurs programmes la calculant.
- La complexité d’une fonction est le $\min$ des complexités des programmes.

- `tri insertion` : $O(n^2)$.
- `quicksort` : $O(n \log(n))$.
- Tri : $O(n \log(n))$.

Complexité implicite

- À chaque programme, sa complexité.
- À chaque programme, sa fonction calculée.
- La complexité de la fonction peut être plus faible que celle du programme.

Complexité implicite

- À chaque programme, sa complexité.
- À chaque programme, sa fonction calculée.
- La complexité de la fonction peut être plus faible que celle du programme.

- `tri insertion` : $O(n^2)$, tri : $O(n \log(n))$.
- Complexité *explicite* : $O(n^2)$.
- Complexité *implicite* : $O(n \log(n))$.

Programmes Primitifs Récursifs

La plus petite classe de programme contenant les constructeurs, les projections, la conditionnelle et close par composition et récursion primitive :

- $\mathtt{f}(\mathbf{Z}, y_1, \cdots, y_n) \rightarrow \mathtt{h}(y_1, \cdots, y_n)$
- $\mathtt{f}(\mathbf{S}(x), y_1, \cdots, y_n) \rightarrow \mathtt{g}(x, y_1, \cdots, y_n, \mathtt{f}(x, y_1, \cdots, y_n))$

PR est la classe de fonctions calculées par des programmes primitifs récursifs.

PR et intensionalité

$$\mathtt{min}(\mathbf{S}(x), \mathbf{Z}) \to \mathbf{Z}$$

$$\mathtt{min}(\mathbf{Z}, \mathbf{S}(y)) \to \mathbf{Z}$$

$$\mathtt{min}(\mathbf{S}(x), \mathbf{S}(y)) \to \mathbf{S}(\mathtt{min}(x, y))$$

Ce programme n'est pas primitif récursif, mais la fonction calculée est dans PR.

PR et intensionalité

$$\texttt{min}(\mathbf{S}(x), \mathbf{Z}) \to \mathbf{Z}$$
$$\texttt{min}(\mathbf{Z}, \mathbf{S}(y)) \to \mathbf{Z}$$
$$\texttt{min}(\mathbf{S}(x), \mathbf{S}(y)) \to \mathbf{S}(\texttt{min}(x, y))$$

Ce programme n'est pas primitif récursif, mais la fonction calculée est dans PR.

Il n'existe aucun algorithme primitif récursif, *de complexité* $O(\min(x, y))$ qui calcule cette fonction (Colson, 89).

PR et intensionalité

$$\mathtt{min}(\mathbf{S}(x), \mathbf{Z}) \rightarrow \mathbf{Z}$$

$$\mathtt{min}(\mathbf{Z}, \mathbf{S}(y)) \rightarrow \mathbf{Z}$$

$$\mathtt{min}(\mathbf{S}(x), \mathbf{S}(y)) \rightarrow \mathbf{S}(\mathtt{min}(x, y))$$

Ce programme n'est pas primitif récursif, mais la fonction calculée est dans PR.

Il n'existe aucun algorithme primitif récursif, *de complexité* $O(\min(x, y))$ qui calcule cette fonction (Colson, 89).

Les algorithmes primitifs récursifs caractérisent *extensionnellement* la classe de fonctions PR, mais la complétude *intensionnelle* est loin.

Récursion bornée

(Cobham, 65)

La plus petite classe de programme contenant les constructeurs (binaires), les projections, la conditionnelle, la fonction *smash* et close par composition et récursion primitive bornée :

- $x\#y = 2^{|x|\cdot|y|}$
- $\mathtt{f}(\mathbf{Z}, y_1, \cdots, y_n) \rightarrow \mathtt{h}(y_1, \cdots, y_n)$
- $\mathtt{f}(\mathbf{S}_i(x), y_1, \cdots, y_n) \rightarrow \mathtt{g}_i(x, y_1, \cdots, y_n, \mathtt{f}(x, y_1, \cdots, y_n))$
  Si $\mathtt{f}(x, y_1, \cdots, y_n) \leq \mathtt{j}(x, y_1, \cdots, y_n)$.

BPR est la classe de fonction calculées par des programmes primitifs récursifs bornés.

Récursion bornée

$$\boxed{\mathrm{BPR} \equiv \mathrm{P\textsc{time}}}$$

L'ensemble des fonctions calculable par un programme primitif récursif borné est exactement PTIME.

Récursion bornée

$$\boxed{\text{BPR} \equiv \text{PTIME}}$$

L'ensemble des fonctions calculable par un programme primitif récursif borné est exactement PTIME.

- Complétude extensionnelle (toutes les *fonctions* sont capturées).
- Pas de complétude intensionnelle (tous les *algorithmes* ne sont pas capturés (`min` de Colson)).

Récursion bornée

$$\boxed{\text{BPR} \equiv \text{P}\textsc{time}}$$

L'ensemble des fonctions calculable par un programme primitif récursif borné est exactement PTIME.

- Complétude extensionnelle (toutes les *fonctions* sont capturées).
- Pas de complétude intensionnelle (tous les *algorithmes* ne sont pas capturés (`min` de Colson)).
- Caractérisation *explicite* (borne fournie avec le programme).

Ordres de terminaison

Ordres de terminaison

(Dershowitz, Mana & Ness, Kamin & Levy, Jouannaud)

Ordre sur les termes (clos) monotone et bien-fondé.

Ordres de terminaison

(Dershowitz, Mana & Ness, Kamin & Levy, Jouannaud)

Ordre sur les termes (clos) monotone et bien-fondé.

Un programme admet un ordre de terminaison si pour chaque règle $l \to r$, on a $r \prec l$.
Dans ce cas, le programme termine (pour toute entrée).

Ordres de terminaison

(Dershowitz, Mana & Ness, Kamin & Levy, Jouannaud)

Ordre sur les termes (clos) monotone et bien-fondé.

Un programme admet un ordre de terminaison si pour chaque règle $l \rightarrow r$, on a $r \prec l$.
Dans ce cas, le programme termine (pour toute entrée).

En effet, toute réduction fait diminuer un sous-terme (règles ordonnées), donc le terme (monotonicité). Le nombre de réductions est donc fini (nœtherianité).

Recursive Path Ordering

(Dershowitz)

$$t = \mathtt{f}(t_1, \cdots, t_n) \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) = s$$

Recursive Path Ordering

(Dershowitz)

$$t = \mathtt{f}(t_1, \cdots, t_n) \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) = s$$

$$\frac{\exists i, t \preceq_{rpo} s_i}{t \prec_{rpo} s}$$

Recursive Path Ordering

(Dershowitz)

$$t = \mathtt{f}(t_1, \cdots, t_n) \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) = s$$

$<_{\mathcal{F}}$ est un ordre sur $\mathcal{F} \bigcup \mathcal{C}$.

$$\frac{\exists i, t \preceq_{rpo} s_i}{t \prec_{rpo} s}$$

$$\frac{f <_{\mathcal{F}} \mathtt{g}}{}$$

Recursive Path Ordering

(Dershowitz)

$$t = \mathtt{f}(t_1, \cdots, t_n) \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) = s$$

$<_{\mathcal{F}}$ est un ordre sur $\mathcal{F} \bigcup \mathcal{C}$.

$$\frac{\exists i, t \preceq_{rpo} s_i}{t \prec_{rpo} s}$$

$$\frac{\forall i, t_i \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) \quad f <_{\mathcal{F}} \mathtt{g}}{}$$

Recursive Path Ordering

(Dershowitz)

$$t = \mathtt{f}(t_1, \cdots, t_n) \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) = s$$

$<_{\mathcal{F}}$ est un ordre sur $\mathcal{F} \bigcup \mathcal{C}$.

$$\frac{\exists i, t \preceq_{rpo} s_i}{t \prec_{rpo} s}$$

$$\frac{\forall i, t_i \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) \quad f <_{\mathcal{F}} \mathtt{g}}{t \prec_{rpo} s}$$

Recursive Path Ordering

(Dershowitz)

$$t = \mathtt{f}(t_1, \cdots, t_n) \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) = s$$

$<_{\mathcal{F}}$ est un ordre sur $\mathcal{F} \bigcup \mathcal{C}$.

$$\frac{\exists i, t \preceq_{rpo} s_i}{t \prec_{rpo} s}$$

$$\frac{\forall i, t_i \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) \quad f <_{\mathcal{F}} \mathtt{g}}{t \prec_{rpo} s}$$

$$\frac{\mathtt{f} \approx_{\mathcal{F}} \mathtt{g}}{}$$

Recursive Path Ordering

(Dershowitz)

$$t = \mathtt{f}(t_1, \cdots, t_n) \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) = s$$

$<_{\mathcal{F}}$ est un ordre sur $\mathcal{F} \bigcup \mathcal{C}$.

$$\frac{\exists i, t \preceq_{rpo} s_i}{t \prec_{rpo} s}$$

$$\frac{\forall i, t_i \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) \quad f <_{\mathcal{F}} \mathtt{g}}{t \prec_{rpo} s}$$

$$\frac{\forall i, t_i \prec_{rpo} s \quad \{t_1, \cdots, t_n\} \prec_{rpo}^{r} \{s_1, \cdots, s_n\} \quad \mathtt{f} \approx_{\mathcal{F}} \mathtt{g}}{}$$

Recursive Path Ordering

(Dershowitz)

$$t = \mathtt{f}(t_1, \cdots, t_n) \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) = s$$

$<_{\mathcal{F}}$ est un ordre sur $\mathcal{F} \bigcup \mathcal{C}$.

$$\frac{\exists i, t \preceq_{rpo} s_i}{t \prec_{rpo} s}$$

$$\frac{\forall i, t_i \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) \quad f <_{\mathcal{F}} \mathtt{g}}{t \prec_{rpo} s}$$

$$\frac{\forall i, t_i \prec_{rpo} s \quad \{t_1, \cdots, t_n\} \prec_{rpo}^{r} \{s_1, \cdots, s_n\} \quad \mathtt{f} \approx_{\mathcal{F}} \mathtt{g}}{t \prec_{rpo} s}$$

MPO, LPO, PPO

- MPO : ordre multi-ensemble.

MPO, LPO, PPO

- MPO : ordre multi-ensemble.
- LPO : ordre lexicographique.
  - $\forall i < j, t_i \preceq_{lpo} s_i$.
  - $t_j \prec_{lpo} s_j$.

MPO, LPO, PPO

- MPO : ordre multi-ensemble.

- LPO : ordre lexicographique.
  - $\forall i < j, t_i \preceq_{lpo} s_i$.
  - $t_j \prec_{lpo} s_j$.

- PPO : ordre produit.
  - $\forall i, t_i \preceq_{ppo} s_{\pi(i)}$.
  - $\exists j, t_j \prec_{ppo} s_{\pi(j)}$.

Caractérisations extensionnelles

$$\boxed{\text{PPO} \equiv \text{MPO} \equiv \text{PR}}$$

(Hofbauer, 92, MM, 00)

L'ensemble des fonctions calculée par des systèmes de réécriture terminant par MPO (PPO) est exactement l'ensemble des fonctions primitives récursives.

Caractérisations extensionnelles

$$\boxed{\text{PPO} \equiv \text{MPO} \equiv \text{PR}}$$

(Hofbauer, 92, MM, 00)

L'ensemble des fonctions calculée par des systèmes de réécriture terminant par MPO (PPO) est exactement l'ensemble des fonctions primitives récursives.

$$\boxed{\text{LPO} \equiv \textsc{MultRec}}$$

(Weiermann 95)

Caractérisations extensionnelles

$$\boxed{\text{PPO} \equiv \text{MPO} \equiv \text{PR}}$$

(Hofbauer, 92, MM, 00)

L'ensemble des fonctions calculée par des systèmes de réécriture terminant par MPO (PPO) est exactement l'ensemble des fonctions primitives récursives.

$$\boxed{\text{LPO} \equiv \textsc{MultRec}}$$

(Weiermann 95)

- Plus intensionnel que PR (`min` de Colson).
- Pas de complétude intensionnelle (`quicksort`).

Analyse prédicative

Séparation des données

$$\begin{aligned}
\mathsf{db}(\mathbf{Z}) &\longrightarrow \mathbf{Z}\\
\mathsf{db}(\mathbf{S}(x)) &\longrightarrow \mathbf{S}(\mathbf{S}(\mathsf{db}(x)))\\
\mathsf{exp}(\mathbf{Z}) &\longrightarrow \mathbf{S}(\mathbf{Z})\\
\mathsf{exp}(\mathbf{S}(x)) &\longrightarrow \mathsf{db}(\mathsf{exp}(x))
\end{aligned}$$

Séparation des données

$$\begin{aligned}
\mathrm{db}(\mathbf{Z}) &\longrightarrow \mathbf{Z}\\
\mathrm{db}(\mathbf{S}(x)) &\longrightarrow \mathbf{S}(\mathbf{S}(\mathrm{db}(x)))\\
\mathrm{exp}(\mathbf{Z}) &\longrightarrow \mathbf{S}(\mathbf{Z})\\
\mathrm{exp}(\mathbf{S}(x)) &\longrightarrow \mathrm{db}(\mathrm{exp}(x))
\end{aligned}$$

Il y a un problème si on permet à un résultat ($\mathrm{exp}(x)$) de contrôller une récursion (db).

Séparation des données

$$\begin{aligned} \mathrm{db}(\mathbf{Z}) &\to \mathbf{Z} \\ \mathrm{db}(\mathbf{S}(x)) &\to \mathbf{S}(\mathbf{S}(\mathrm{db}(x))) \\ \mathrm{exp}(\mathbf{Z}) &\to \mathbf{S}(\mathbf{Z}) \\ \mathrm{exp}(\mathbf{S}(x)) &\to \mathrm{db}(\mathrm{exp}(x)) \end{aligned}$$

Il y a un problème si on permet à un résultat ($\mathrm{exp}(x)$) de contrôller une récursion (db).

- Donner à chaque terme de l'énergie (valence, arguments sûrs/normaux).
- Une récursion se fait sur un argument normal (de valence 1, avec de l'énergie).
- Une récursion consomme de l'énergie : le résultat est sûr (de valence 0).

Récursion primitive sûre

(Bellantoni & Cook, 92)

On sépare les arguments en arguments *normaux* et *sûrs* :

$$\mathtt{f}(x_1, \cdots, x_n \, ; \, y_1, \cdots, y_m)$$

Récursion primitive sûre

(Bellantoni & Cook, 92)

On sépare les arguments en arguments *normaux* et *sûrs* :

$$\mathtt{f}(x_1, \cdots, x_n \,;\, y_1, \cdots, y_m)$$

La classe B est la plus petite classe de programme contenant les constructeurs (binaires), les projections, la conditionnelle et close par composition sûre et récursion primitive sûre :

Récursion primitive sûre

(Bellantoni & Cook, 92)

On sépare les arguments en arguments *normaux* et *sûrs* :

$$\mathtt{f}(x_1, \cdots, x_n \,;\, y_1, \cdots, y_m)$$

La classe B est la plus petite classe de programme contenant les constructeurs (binaires), les projections, la conditionnelle et close par composition sûre et récursion primitive sûre :

- $\mathtt{f}(\mathbf{Z}, x_1, \cdots, x_n \,;\, y_1, \cdots, y_m) \to \mathtt{h}(x_1, \cdots, x_n \,;\, y_1, \cdots, y_m)$
- $\mathtt{f}(\mathbf{S}_i(x_0), x_1, \cdots, x_n \,;\, y_1, \cdots, y_m) \to \mathtt{g}_i(x_0, x_1, \cdots, x_n \,;\, y_1, \cdots, y_m, \mathtt{f}(x_0, x_1, \cdots, x_n ; y_1, \cdots, y_m))$

Récursion primitive sûre

(Bellantoni & Cook, 92)

On sépare les arguments en arguments *normaux* et *sûrs* :

$$\mathtt{f}(x_1, \cdots, x_n \,;\, y_1, \cdots, y_m)$$

La classe B est la plus petite classe de programme contenant les constructeurs (binaires), les projections, la conditionnelle et close par composition sûre et récursion primitive sûre :

- $\mathtt{f}(\mathbf{Z}, x_1, \cdots, x_n \,;\, y_1, \cdots, y_m) \to \mathtt{h}(x_1, \cdots, x_n \,;\, y_1, \cdots, y_m)$
- $\mathtt{f}(\mathbf{S}_i(x_0), x_1, \cdots, x_n \,;\, y_1, \cdots, y_m) \to$ $\mathtt{g}_i(x_0, x_1, \cdots, x_n \,;\, y_1, \cdots, y_m, \mathtt{f}(x_0, x_1, \cdots, x_n ; y_1, \cdots, y_m))$
- $\mathtt{f}(x_1, \cdots, x_n \,;\, y_1, \cdots, y_m) \to$ $\mathtt{h}(\mathtt{r}(x_1, \cdots, x_n \,;) \,;\, \mathtt{g}(x_1, \cdots, x_n \,;\, y_1, \cdots, y_m))$

Caractérisation extensionnelle

$$\boxed{B \equiv \textsc{Ptime}}$$

- Caractérisation implicite (plus de borne explicite).
- Complétude extensionnelle.
- Mauvaise intensionalité.

Light MPO

(Marion, 00)

$$B = PR + \text{ tiering } \equiv \mathrm{P}\textsc{time}$$

$$MPO \equiv PR$$

Light MPO

(Marion, 00)

$$B = PR + \text{ tiering } \equiv \text{PTIME}$$

$$MPO \equiv PR$$

$$MPO + \text{ tiering } \equiv \text{PTIME}?$$

Light MPO

(Marion, 00)

$$B = PR + \text{ tiering } \equiv \text{PTIME}$$

$$MPO \equiv PR$$

$$MPO + \text{ tiering } \equiv \text{PTIME}?$$

- Valence (0 ou 1) pour les arguments (resp. sûrs ou normaux).
- La comparaison doit respecter les valences.
- Comparaison produit, la décroissance doit avoir lieu sur un argument de valence 1.

LMPO

$$\frac{s \preceq_k t_i}{s \prec_k \mathtt{c}(\ldots, t_i, \ldots)}\ k \in \{0,1\}$$

$$\frac{s \preceq_{\nu(\mathtt{f},i)} t_i}{s \prec_k \mathtt{f}(\ldots, t_i, \ldots)}\ k \leq \nu(\mathtt{f}, i)$$

$$\frac{s_i \prec_{max(k,\nu(f,i))} \mathtt{g}(t_1, \cdots, t_n) \quad f <_{\mathcal{F}} \mathtt{g}}{f(s_1, \cdots, s_m) \prec_k \mathtt{g}(t_1, \cdots, t_n)}$$

$$\frac{\{s_1, \cdots, s_n\} \prec^p_{\mathtt{g},\mathtt{f}} \{t_1, \cdots, t_n\} \quad \mathtt{f} \approx_{\mathcal{F}} \mathtt{g}}{\mathtt{g}(s_1, \cdots, s_n) \prec_{\mathbf{0}} \mathtt{f}(t_1, \cdots, t_n)}$$

Longest Common Subsequence

$$AABBA$$

$$BABAABA$$

Longest Common Subsequence

$$AABBA$$

$$BABAABA$$

Longest Common Subsequence

$$AABBA$$

$$BABAABA$$

Longest Common Subsequence

$\mathtt{lcs} : \mathsf{Word}_1, \mathsf{Word}_1 \longrightarrow \mathsf{Nat}$

$$\mathtt{lcs}(x, \epsilon\ ;) \rightarrow \mathbf{Z}$$

$$\mathtt{lcs}(\epsilon, y\ ;) \rightarrow \mathbf{Z}$$

$$\mathtt{lcs}(\mathbf{i}(x), \mathbf{i}(y)\ ;) \rightarrow \mathbf{S}(\mathtt{lcs}(x, y\ ;))$$

$$\mathtt{lcs}(\mathbf{i}(x), \mathbf{j}(y)\ ;) \rightarrow \mathtt{max}(\qquad ; \mathtt{lcs}(x, \mathbf{j}(y)\ ;), \mathtt{lcs}(\mathbf{i}(x), y\ ;))$$

Longest Common Subsequence

$\mathtt{max} : \quad \mathsf{Nat}_0, \mathsf{Nat}_0 \to \mathsf{Nat}$

$$\mathtt{max}(\ ; \mathbf{Z}, n) \to n$$

$$\mathtt{max}(\ ; m, \mathbf{Z}) \to m$$

$$\mathtt{max}(\ ; \mathbf{S}(m), \mathbf{S}(n)) \to \mathbf{S}(\mathtt{max}(\ ; m, n))$$

$\mathtt{lcs} : \mathsf{Word}_1, \mathsf{Word}_1 \to \mathsf{Nat}$

$$\mathtt{lcs}(x, \epsilon\ ;) \to \mathbf{Z}$$

$$\mathtt{lcs}(\epsilon, y\ ;) \to \mathbf{Z}$$

$$\mathtt{lcs}(\mathbf{i}(x), \mathbf{i}(y)\ ;) \to \mathbf{S}(\mathtt{lcs}(x, y\ ;))$$

$$\mathtt{lcs}(\mathbf{i}(x), \mathbf{j}(y)\ ;) \to \mathtt{max}(\ ; \mathtt{lcs}(x, \mathbf{j}(y)\ ;), \mathtt{lcs}(\mathbf{i}(x), y\ ;))$$

Longest Common Subsequence

$\mathtt{max} \;:\; \text{Word}_1,\; \text{Nat}_0, \text{Nat}_0 \rightarrow \text{Nat}$

$$\mathtt{max}(x\;;\;\mathbf{Z}, n) \rightarrow n$$

$$\mathtt{max}(x\;;\;m, \mathbf{Z}) \rightarrow m$$

$$\mathtt{max}(\mathbf{i}(x)\;;\;\mathbf{S}(m), \mathbf{S}(n)) \rightarrow \mathbf{S}(\mathtt{max}(x\;;\;m, n))$$

$\mathtt{lcs} : \text{Word}_1, \text{Word}_1 \rightarrow \text{Nat}$

$$\mathtt{lcs}(x, \epsilon\;;) \rightarrow \mathbf{Z}$$

$$\mathtt{lcs}(\epsilon, y\;;) \rightarrow \mathbf{Z}$$

$$\mathtt{lcs}(\mathbf{i}(x), \mathbf{i}(y)\;;) \rightarrow \mathbf{S}(\mathtt{lcs}(x, y\;;))$$

$$\mathtt{lcs}(\mathbf{i}(x), \mathbf{j}(y)\;;) \rightarrow \mathtt{max}(\mathbf{i}(x)\;;\;\mathtt{lcs}(x, \mathbf{j}(y)\;;), \mathtt{lcs}(\mathbf{i}(x), y\;;))$$

Complexité implicite

$$u = abaaab \qquad v = aabb$$

$$\texttt{lcs}(\epsilon, y) \rightarrow \mathsf{Z}$$

$$\texttt{lcs}(x, \epsilon) \rightarrow \mathsf{Z}$$

$$\texttt{lcs}(\mathsf{i}(x), \mathsf{i}(y)) \rightarrow \mathsf{S}(\texttt{lcs}(x, y))$$

$$\texttt{lcs}(\mathsf{i}(x), \mathsf{j}(y)) \rightarrow max(\texttt{lcs}(\mathsf{i}(x), y), \texttt{lcs}(x, \mathsf{j}(y)))$$

Complexité implicite

$$u = abaaab \qquad v = aabb$$

$$\begin{aligned}
\mathtt{lcs}(\epsilon, y) &\rightarrow \mathbf{Z} \\
\mathtt{lcs}(x, \epsilon) &\rightarrow \mathbf{Z} \\
\mathtt{lcs}(\mathbf{i}(x), \mathbf{i}(y)) &\rightarrow \mathbf{S}(\mathtt{lcs}(x, y)) \\
\mathtt{lcs}(\mathbf{i}(x), \mathbf{j}(y)) &\rightarrow max(\mathtt{lcs}(\mathbf{i}(x), y), \mathtt{lcs}(x, \mathbf{j}(y)))
\end{aligned}$$

Complexité *explicite* : exponentielle.

Complexité implicite

$$u = abaaab \qquad v = aabb$$

$$\begin{aligned}
\mathtt{lcs}(\epsilon, y) &\rightarrow \mathbf{Z} \\
\mathtt{lcs}(x, \epsilon) &\rightarrow \mathbf{Z} \\
\mathtt{lcs}(\mathbf{i}(x), \mathbf{i}(y)) &\rightarrow \mathbf{S}(\mathtt{lcs}(x, y)) \\
\mathtt{lcs}(\mathbf{i}(x), \mathbf{j}(y)) &\rightarrow max(\mathtt{lcs}(\mathbf{i}(x), y), \mathtt{lcs}(x, \mathbf{j}(y)))
\end{aligned}$$

Complexité *explicite* : exponentielle.
Termine par LMPO.
Complexité *implicite* : polynômiale (programmation dynamique).

Mémoïsation

La complexité exponentielle vient de calculs qui sont faits plusieurs fois.

Mémoïsation

La complexité exponentielle vient de calculs qui sont faits plusieurs fois.

$$lcs(aa, bb)$$

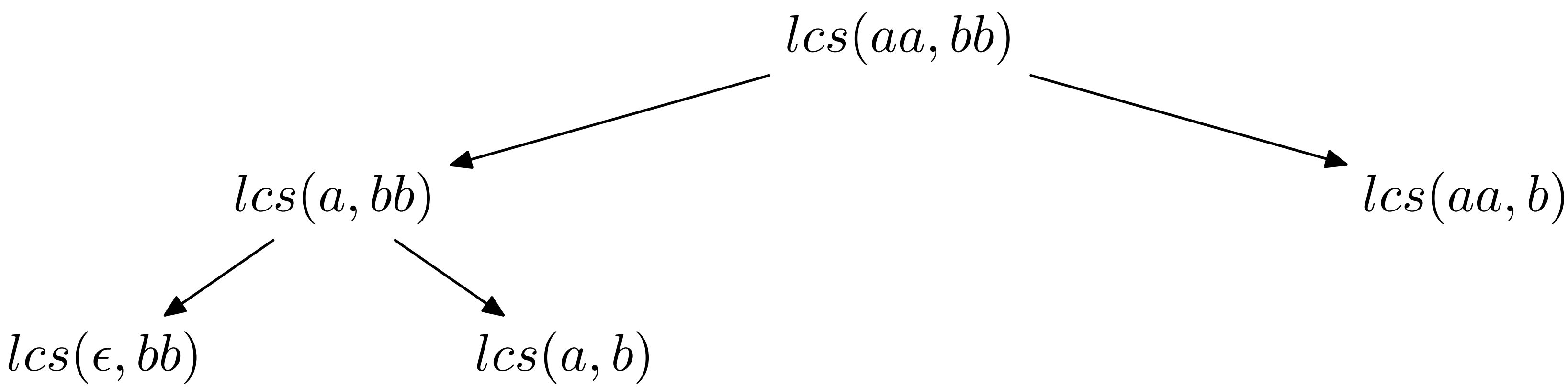
Mémoïsation

La complexité exponentielle vient de calculs qui sont faits plusieurs fois.

$$\begin{tikzcd} & lcs(aa, bb) \arrow[dl] \arrow[dr] & \\ lcs(a, bb) & & lcs(aa, b) \end{tikzcd}$$

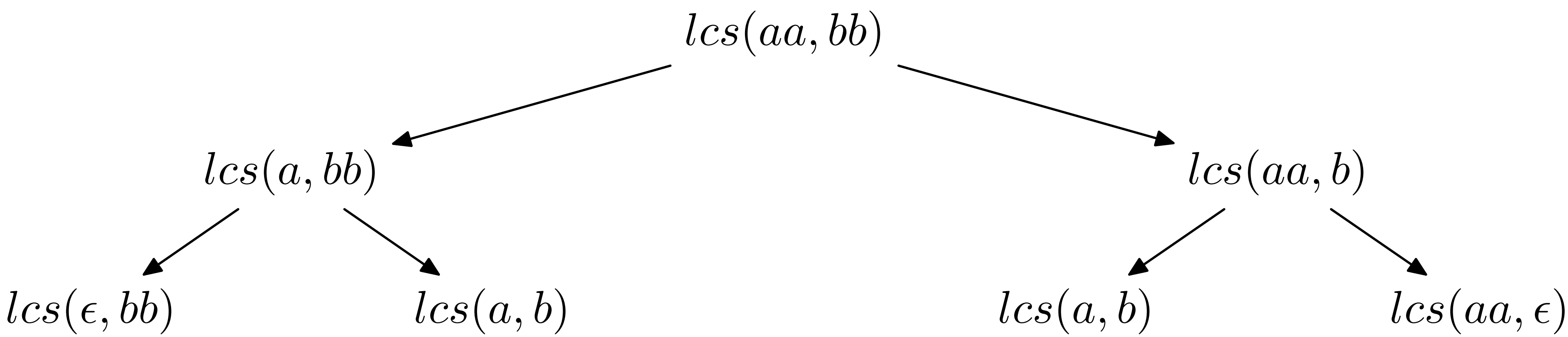
Mémoïsation

La complexité exponentielle vient de calculs qui sont faits plusieurs fois.



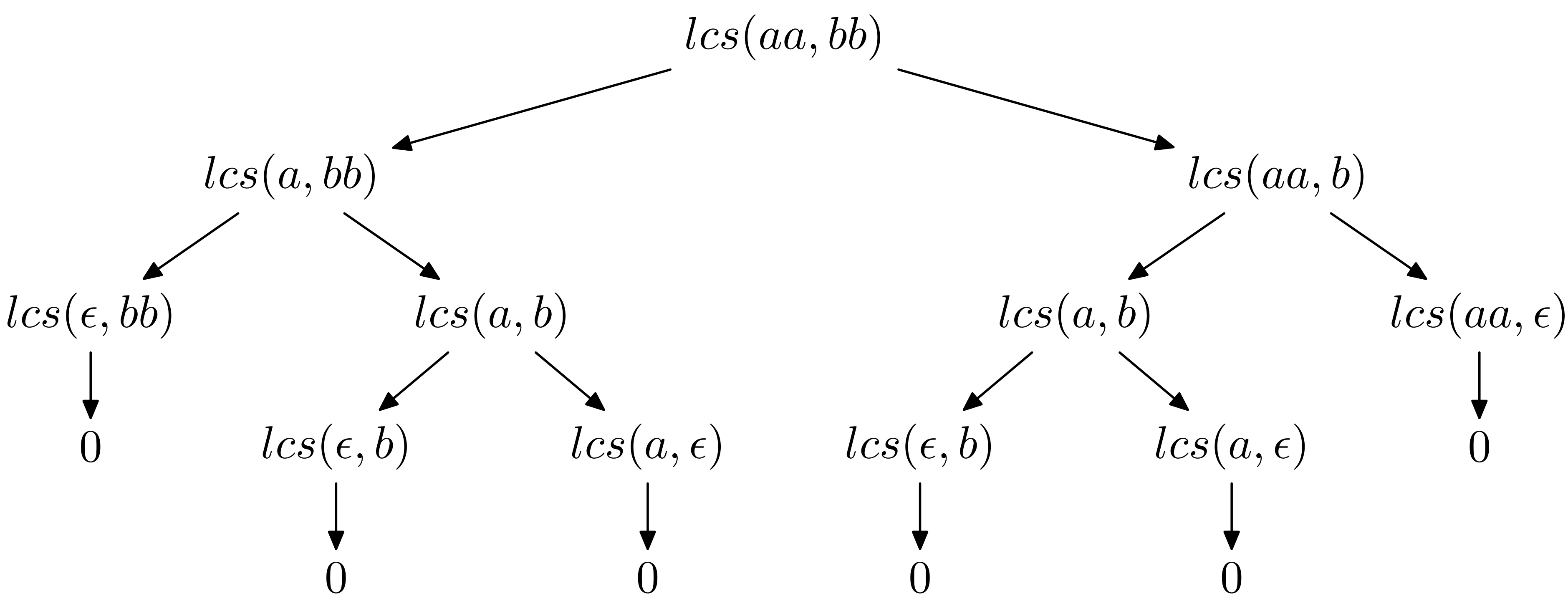
Mémoïsation

La complexité exponentielle vient de calculs qui sont faits plusieurs fois.



Mémoïsation

La complexité exponentielle vient de calculs qui sont faits plusieurs fois.

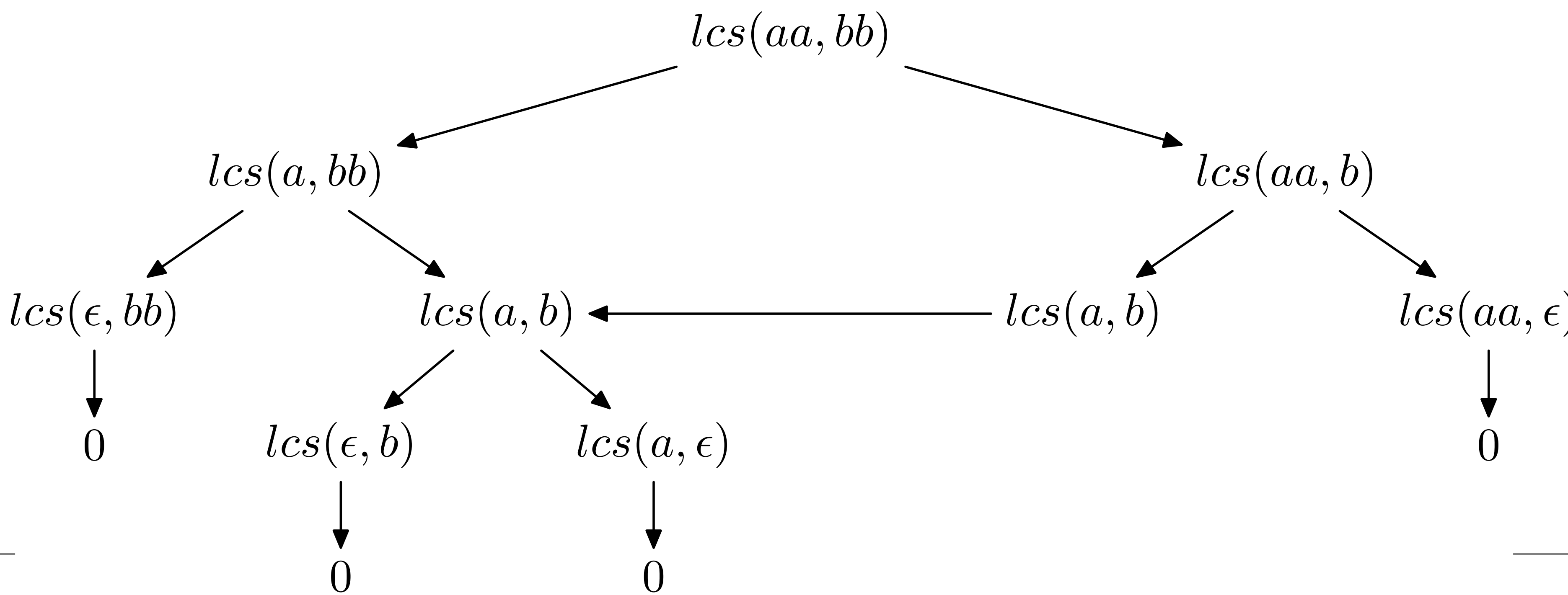


Mémoïsation

La complexité exponentielle vient de calculs qui sont faits plusieurs fois.
Pour atteindre la borne polynômiale, on garde en mémoire les résultats des calculs. Si on a de nouveau besoin du même résultat, il suffit d'aller le chercher en mémoire.

Mémoïsation

La complexité exponentielle vient de calculs qui sont faits plusieurs fois.
Pour atteindre la borne polynômiale, on garde en mémoire les résultats des calculs. Si on a de nouveau besoin du même résultat, il suffit d'aller le chercher en mémoire.



Mémoïsation

La complexité exponentielle vient de calculs qui sont faits plusieurs fois.
Pour atteindre la borne polynômiale, on garde en mémoire les résultats des calculs. Si on a de nouveau besoin du même résultat, il suffit d'aller le chercher en mémoire.
Le cache peut être minimisé grace aux informations données par LMPO (Marion, 2000).

Interpréteur avec cache

$$\frac{}{\mathcal{E}, \sigma \vdash \langle C, x \rangle}$$

Interpréteur avec cache

$$\frac{\sigma(x) = v}{\mathcal{E}, \sigma \vdash \langle C, x \rangle}$$

Interpréteur avec cache

$$\frac{\sigma(x) = v}{\mathcal{E}, \sigma \vdash \langle C, x\rangle \rightarrow \langle C, v\rangle}$$

Interpréteur avec cache

$$\frac{\sigma(x) = v}{\mathcal{E}, \sigma \vdash \langle C, x \rangle \rightarrow \langle C, v \rangle} \qquad \frac{\mathbf{c} \in \mathcal{C}}{\mathcal{E}, \sigma \vdash \langle C_0, \mathbf{c}(\vec{t}) \rangle}$$

Interpréteur avec cache

$$\frac{\sigma(x) = v}{\mathcal{E}, \sigma \vdash \langle C, x \rangle \longrightarrow \langle C, v \rangle} \qquad \frac{\mathbf{c} \in \mathcal{C} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle}{\mathcal{E}, \sigma \vdash \langle C_0, \mathbf{c}(\vec{t}) \rangle}$$

Interpréteur avec cache

$$\frac{\sigma(x) = v}{\mathcal{E}, \sigma \vdash \langle C, x \rangle \longrightarrow \langle C, v \rangle} \qquad \frac{\mathbf{c} \in \mathcal{C} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle}{\mathcal{E}, \sigma \vdash \langle C_0, \mathbf{c}(\vec{t}) \rangle \longrightarrow \langle C_n, \mathbf{c}(\vec{v}) \rangle}$$

Interpréteur avec cache

$$\frac{\sigma(x) = v}{\mathcal{E}, \sigma \vdash \langle C, x \rangle \longrightarrow \langle C, v \rangle} \qquad \frac{\mathbf{c} \in \mathcal{C} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle}{\mathcal{E}, \sigma \vdash \langle C_0, \mathbf{c}(\vec{t}) \rangle \longrightarrow \langle C_n, \mathbf{c}(\vec{v}) \rangle}$$

$$\frac{\mathtt{f} \in \mathcal{F} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{f}(\vec{t}) \rangle}$$

Interpréteur avec cache

$$\frac{\sigma(x) = v}{\mathcal{E}, \sigma \vdash \langle C, x \rangle \longrightarrow \langle C, v \rangle} \qquad \frac{\mathbf{c} \in \mathcal{C} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle}{\mathcal{E}, \sigma \vdash \langle C_0, \mathbf{c}(\vec{t}) \rangle \longrightarrow \langle C_n, \mathbf{c}(\vec{v}) \rangle}$$

$$\frac{\mathtt{f} \in \mathcal{F} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle \quad (\mathtt{f}(\vec{v}), v) \in C_n}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{f}(\vec{t}) \rangle}$$

Interpréteur avec cache

$$\frac{\sigma(x) = v}{\mathcal{E}, \sigma \vdash \langle C, x \rangle \longrightarrow \langle C, v \rangle} \qquad \frac{\mathbf{c} \in \mathcal{C} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle}{\mathcal{E}, \sigma \vdash \langle C_0, \mathbf{c}(\vec{t}) \rangle \longrightarrow \langle C_n, \mathbf{c}(\vec{v}) \rangle}$$

$$\frac{\mathtt{f} \in \mathcal{F} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle \quad (\mathtt{f}(\vec{v}), v) \in C_n}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{f}(\vec{t}) \rangle \longrightarrow \langle C_n, v \rangle}$$

Interpréteur avec cache

$$\frac{\sigma(x) = v}{\mathcal{E}, \sigma \vdash \langle C, x\rangle \longrightarrow \langle C, v\rangle} \qquad \frac{\mathtt{c} \in \mathcal{C} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i\rangle \longrightarrow \langle C_i, v_i\rangle}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{c}(\vec{t})\rangle \longrightarrow \langle C_n, \mathtt{c}(\vec{v})\rangle}$$

$$\frac{\mathtt{f} \in \mathcal{F} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i\rangle \longrightarrow \langle C_i, v_i\rangle \quad (\mathtt{f}(\vec{v}), v) \in C_n}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{f}(\vec{t})\rangle \longrightarrow \langle C_n, v\rangle}$$

$$\frac{\mathtt{f} \in \mathcal{F} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i\rangle \longrightarrow \langle C_i, v_i\rangle}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{f}(\vec{t})\rangle}$$

Interpréteur avec cache

$$\frac{\sigma(x) = v}{\mathcal{E}, \sigma \vdash \langle C, x \rangle \longrightarrow \langle C, v \rangle} \qquad \frac{\mathtt{c} \in \mathcal{C} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{c}(\vec{t}) \rangle \longrightarrow \langle C_n, \mathtt{c}(\vec{v}) \rangle}$$

$$\frac{\mathtt{f} \in \mathcal{F} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle \quad (\mathtt{f}(\vec{v}), v) \in C_n}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{f}(\vec{t}) \rangle \longrightarrow \langle C_n, v \rangle}$$

$$\frac{\begin{array}{c}\mathtt{f} \in \mathcal{F} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle \\ \mathtt{f}(\vec{p}) \rightarrow r \in \mathcal{E} \quad p_i\sigma' = v_i\end{array}}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{f}(\vec{t}) \rangle}$$

Interpréteur avec cache

$$\frac{\sigma(x) = v}{\mathcal{E}, \sigma \vdash \langle C, x \rangle \longrightarrow \langle C, v \rangle} \qquad \frac{\mathtt{c} \in \mathcal{C} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{c}(\vec{t}) \rangle \longrightarrow \langle C_n, \mathtt{c}(\vec{v}) \rangle}$$

$$\frac{\mathtt{f} \in \mathcal{F} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle \quad (\mathtt{f}(\vec{v}), v) \in C_n}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{f}(\vec{t}) \rangle \longrightarrow \langle C_n, v \rangle}$$

$$\frac{\begin{array}{c}\mathtt{f} \in \mathcal{F} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle \\ \mathtt{f}(\vec{p}) \rightarrow r \in \mathcal{E} \quad p_i\sigma' = v_i \quad \mathcal{E}, \sigma' \vdash \langle C_n, r \rangle \longrightarrow \langle C, v \rangle\end{array}}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{f}(\vec{t}) \rangle}$$

Interpréteur avec cache

$$\frac{\sigma(x) = v}{\mathcal{E}, \sigma \vdash \langle C, x \rangle \longrightarrow \langle C, v \rangle} \qquad \frac{\mathtt{c} \in \mathcal{C} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{c}(\vec{t}) \rangle \longrightarrow \langle C_n, \mathtt{c}(\vec{v}) \rangle}$$

$$\frac{\mathtt{f} \in \mathcal{F} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle \quad (\mathtt{f}(\vec{v}), v) \in C_n}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{f}(\vec{t}) \rangle \longrightarrow \langle C_n, v \rangle}$$

$$\frac{\begin{array}{c}\mathtt{f} \in \mathcal{F} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \longrightarrow \langle C_i, v_i \rangle \\ \mathtt{f}(\vec{p}) \rightarrow r \in \mathcal{E} \quad p_i \sigma' = v_i \quad \mathcal{E}, \sigma' \vdash \langle C_n, r \rangle \longrightarrow \langle C, v \rangle\end{array}}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{f}(\vec{t}) \rangle \longrightarrow \langle C \bigcup (\mathtt{f}(\vec{v}), v), v \rangle}$$

Quasi-Interprétations

Interprétations polynomiales

(Lankford, 1979)

Une interprétation polynomiale d'un symbole a est une fonction $[\![a]\!]$ telle que :

Interprétations polynomiales

(Lankford, 1979)

Une interprétation polynomiale d'un symbole a est une fonction $[\![a]\!]$ telle que :

- $[\![a]\!]$ est bornée par un polynôme.

Interprétations polynomiales

(Lankford, 1979)

Une interprétation polynomiale d'un symbole a est une fonction $[\![a]\!]$ telle que :

- $[\![a]\!]$ est bornée par un polynôme.
- $[\![\mathbf{c}]\!](X_1, \cdots, X_n) = \sum X_i + c^{\underline{te}}$

Interprétations polynomiales

(Lankford, 1979)

Une interprétation polynomiale d'un symbole a est une fonction $[\![a]\!]$ telle que :

- $[\![a]\!]$ est bornée par un polynôme.
- $[\![\mathbf{c}]\!](X_1, \cdots, X_n) = \sum X_i + c^{\underline{te}}$
- $[\![a]\!](X_1, \cdots, X_n) > \sum X_i$

Interprétations polynomiales

(Lankford, 1979)

Une interprétation polynomiale d'un symbole a est une fonction $[\![a]\!]$ telle que :

- $[\![a]\!]$ est bornée par un polynôme.
- $[\![\mathbf{c}]\!](X_1, \cdots, X_n) = \sum X_i + c^{\underline{te}}$
- $[\![a]\!](X_1, \cdots, X_n) > \sum X_i$
- $[\![a]\!]$ est croissante (strictement).

Interprétations polynomiales

(Lankford, 1979)

Une interprétation polynomiale d'un symbole a est une fonction $[\![a]\!]$ telle que :

- $[\![a]\!]$ est bornée par un polynôme.
- $[\![\mathbf{c}]\!](X_1, \cdots, X_n) = \sum X_i + c^{\underline{te}}$
- $[\![a]\!](X_1, \cdots, X_n) > \sum X_i$
- $[\![a]\!]$ est croissante (strictement).

$$[\![a(t_1, \cdots, t_n)]\!] = [\![a]\!]([\![t_1]\!], \cdots, [\![t_n]\!])$$

Interprétations polynomiales

(Lankford, 1979)

Une interprétation polynomiale d'un symbole a est une fonction $[\![a]\!]$ telle que :

- $[\![a]\!]$ est bornée par un polynôme.
- $[\![\mathbf{c}]\!](X_1, \cdots, X_n) = \sum X_i + c^{\underline{te}}$
- $[\![a]\!](X_1, \cdots, X_n) > \sum X_i$
- $[\![a]\!]$ est croissante (strictement).

$$[\![a(t_1, \cdots, t_n)]\!] = [\![a]\!]([\![t_1]\!], \cdots, [\![t_n]\!])$$

Un système admet une interprétation si pour chaque règle $l \rightarrow r$ on a $[\![r]\!] < [\![l]\!]$.
Le système termine alors en temps polynômial (BCMT, 1998).

Quasi-interprétations polynômiales

(Marion et Moyen, Bonfante, 2000)

Une quasi-interprétation polynômiale d'un symbole a est une fonction $(\!|a|\!)$ telle que :

- $(\!|a|\!)$ est bornée par un polynôme.
- $(\!|\mathbf{c}|\!)(X_1, \cdots, X_n) = \sum X_i + c^{\underline{te}}$
- $(\!|a|\!)(X_1, \cdots, X_n) \geq X_i$ pour tout i.
- $(\!|a|\!)$ est croissante (non-strictement).

$$(\!|a(t_1, \cdots, t_n)|\!) = (\!|a|\!)((\!|t_1|\!), \cdots, (\!|t_n|\!))$$

Un système admet une quasi-interprétation si pour chaque règle $l \to r$ on a $(\!|r|\!) \leq (\!|l|\!)$.

Le système peut ne pas terminer ($\mathtt{f}(x) \to \mathtt{f}(x)$).

Quasi-interprétations et taille

- Termes constructeurs : $(\!|v|\!) \approx |v|$

Quasi-interprétations et taille

- Termes constructeurs : $(\!|v|\!) \approx |v|$
- Réductions : $t \xrightarrow{+} s \Rightarrow (\!|t|\!) \geq (\!|s|\!)$.

Quasi-interprétations et taille

- Termes constructeurs : $(\!|v|\!) \approx |v|$
- Réductions : $t \xrightarrow{+} s \Rightarrow (\!|t|\!) \geq (\!|s|\!)$.

Lemme “Plug and play” :

$$\boxed{\begin{array}{c}\text{Si } \mathtt{f}(v_1, \cdots, v_n) \xrightarrow{!} v, \\ \text{alors } |v| = P(|v_1|, \ldots, |v_n|)\end{array}}$$

Quasi-interprétations et taille

- Termes constructeurs : $(\!|v|\!) \approx |v|$
- Réductions : $t \xrightarrow{+} s \Rightarrow (\!|t|\!) \geq (\!|s|\!)$.

Lemme “Plug and play” :

$$\boxed{\begin{array}{c}\text{Si } \mathtt{f}(v_1, \cdots, v_n) \xrightarrow{!} v, \\ \text{alors } |v| = P(|v_1|, \ldots, |v_n|)\end{array}}$$

$$|v| \approx (\!|v|\!) \leq (\!|\mathtt{f}(v_1, \cdots, v_n)|\!) \leq P((\!|v_1|\!), \ldots, (\!|v_n|\!)) \approx P(|v_1|, \ldots, |v_n|)$$

LCS, encore

$$\mathtt{max}(\mathbf{Z}, n) \to n$$
$$\mathtt{max}(m, \mathbf{Z}) \to m$$
$$\mathtt{max}(\mathbf{S}(m), \mathbf{S}(n)) \to \mathbf{S}(\mathtt{max}(m, n))$$

$$\mathtt{lcs}(x, \epsilon) \to \mathbf{Z}$$
$$\mathtt{lcs}(\epsilon, y) \to \mathbf{Z}$$
$$\mathtt{lcs}(\mathbf{i}(x), \mathbf{i}(y)) \to \mathbf{S}(\mathtt{lcs}(x, y))$$
$$\mathtt{lcs}(\mathbf{i}(x), \mathbf{j}(y)) \to \mathtt{max}(\mathtt{lcs}(x, \mathbf{j}(y)), \mathtt{lcs}(\mathbf{i}(x), y))$$

$(\!|\mathtt{max}|\!)(X, Y) = (\!|\mathtt{lcs}|\!)(X, Y) = \max(X, Y)$
Le programme n'admet pas d'interprétation.

Caractérisation de PTIME

$$\boxed{\text{MPO + QI} \equiv \text{PTIME}}$$

(Marion et Moyen, 2000)

- Complexité implicite (mémoïsation).
- Plus intensionnel que LMPO (`lcs`).
- Non intensionnellement complet (`quicksort`).
- Caractérisation implicite ou explicite ?
- Le système ICAR (Moyen, 2001) permet de faire cette analyse.

Autres caractérisations

LPO + QI $\equiv$ PSPACE

(Bonfante, Marion et Moyen, 2001)

La borne peut être extraite de la preuve de terminaison (Amadio et al. 2004).

LPO + linéarité + QI $\equiv$ PTIME

LPO + MPO + QI $\equiv$ PSPACE

MPO + NonDéterminisme + QI $\equiv$ PSPACE

LPO + QI$_{aff}$ $\equiv$ LINSPACE

(BMM, 05)

Conjectures

$$\boxed{\text{LPO + QI } (\max, +) \overset{?}{\equiv} \text{NLINSPACE}}$$

$$\boxed{\text{terminaison + QI} \overset{?}{\equiv} \text{PSPACE}}$$

(BMM, 05)

Synthèse de QIs

QI et sémantique

Souvent, la sémantique du programme fournit une quasi-interprétation :

$$\mathsf{add}(\mathbf{Z}, y) \to y$$

$$\mathsf{add}(\mathbf{S}(x), y) \to \mathbf{S}(\mathsf{add}(x, y))$$

QI et sémantique

Souvent, la sémantique du programme fournit une quasi-interprétation :

$$\mathsf{add}(\mathbf{Z}, y) \to y$$

$$\mathsf{add}(\mathbf{S}(x), y) \to \mathbf{S}(\mathsf{add}(x, y))$$

- $(\!|\mathsf{add}|\!)(X, Y) = X + Y$

QI et sémantique

Souvent, la sémantique du programme fournit une quasi-interprétation :

$$\mathsf{add}(\mathbf{Z}, y) \rightarrow y$$

$$\mathsf{add}(\mathbf{S}(x), y) \rightarrow \mathbf{S}(\mathsf{add}(x, y))$$

- $(\!|\mathsf{add}|\!)(X, Y) = X + Y$
- $[\![\mathsf{add}]\!](X, Y) = 2X + Y + 1$

Ensemble de définition

- Interprétations polynomiales :

- Quasi-Interprétations :

Ensemble de définition

- Interprétations polynomiales :
  - Ordre de terminaison $\Rightarrow$ bien-fondé.
  - $[\![a]\!] : \mathbb{N}^k \rightarrow \mathbb{N}$

- Quasi-Interprétations :

Ensemble de définition

- Interprétations polynomiales :
  - Ordre de terminaison $\Rightarrow$ bien-fondé.
  - $[\![a]\!] : \mathbb{N}^k \to \mathbb{N}$

- Quasi-Interprétations :
  - Ne sert pas pour la terminaison.
  - $(\!|a|\!) : \mathbb{R}^k \to \mathbb{R}$.

Ensemble de définition

- Interprétations polynomiales :
  - Ordre de terminaison $\Rightarrow$ bien-fondé.
  - $[\![a]\!] : \mathbb{N}^k \to \mathbb{N}$
- Quasi-Interprétations :
  - Ne sert pas pour la terminaison.
  - $(\!|a|\!) : \mathbb{R}^k \to \mathbb{R}$.

Alors que le théorème de Matiyasevich empêche de *vérifier* une interprétation, le théorème de Tarski permet de *synthétiser* une quasi-interprétation.

QIs et Tarski

Théorème de Tarski : l'élimination des quantificateurs sur $(\mathbb{R}, +, \times, \leq)$ est décidable (EXPTIME).

QIs et Tarski

Théorème de Tarski : l'élimination des quantificateurs sur $(\mathbb{R}, +, \times, \leq)$ est décidable (EXPTIME).

$$(\!|t|\!) \text{ peut s'écrire } \max_i \{P_i(X_1, \cdots, X_n)\}$$

$$(\!|t|\!) \geq (\!|s|\!) \Leftrightarrow \forall X_i, \max_i \{P_i(X_1, \cdots, X_n)\} \geq \max_j \{Q_j(X_1, \cdots, X_n)\}$$

QIs et Tarski

Théorème de Tarski : l'élimination des quantificateurs sur $(\mathbb{R}, +, \times, \leq)$ est décidable (EXPTIME).

$$(\!|t|\!) \text{ peut s'écrire } \max_i \{P_i(X_1, \cdots, X_n)\}$$

$$(\!|t|\!) \geq (\!|s|\!) \Leftrightarrow \forall X_i, \max_i \{P_i(X_1, \cdots, X_n)\} \geq \max_j \{Q_j(X_1, \cdots, X_n)\}$$

$$\Leftrightarrow \forall X_i, \bigvee_i \bigwedge_j P_i(X_1, \cdots, X_n) \geq Q_j(X_1, \cdots, X_n)$$

QIs et Tarski

Théorème de Tarski : l'élimination des quantificateurs sur $(\mathbb{R}, +, \times, \leq)$ est décidable (EXPTIME).

$$(\!|t|\!) \text{ peut s'écrire } \max_i \{P_i(X_1, \cdots, X_n)\}$$

$$(\!|t|\!) \geq (\!|s|\!) \Leftrightarrow \forall X_i, \max_i \{P_i(X_1, \cdots, X_n)\} \geq \max_j \{Q_j(X_1, \cdots, X_n)\}$$

$$\Leftrightarrow \forall X_i, \bigvee_i \bigwedge_j P_i(X_1, \cdots, X_n) \geq Q_j(X_1, \cdots, X_n)$$

- Vérifier une quasi-interprétation sur $\mathbb{R}$ est décidable.
- Synthétiser une quasi-interprétation sur $\mathbb{R}$ est décidable à degré borné ($\exists \alpha_k, \forall X_i, \ldots$).

Bornes inférieures

Synthétiser une quasi-interprétation sur $\mathbb{Q}$ ou sur $\mathbb{R}$ est NP-difficile, et NP-complet pour les QIs $(\max, +)$.

(Amadio, 04, Péchoux, 04)

Synthétiser une quasi-interprétation *somme multilinéaire* ($(\!|a|\!)(X_1, \cdots, X_n) = \sum X_i + \alpha$) est PTIME.

(Amadio, 04)

Décider si un programme termine par MPO/LPO est NP-complet.

(Krishnamoorthy et Narendran, 85)

Non Size Increasing

Un programme est NSI si le calcul de $p(x)$ s'effectue en espace $|x| + \alpha$.

Non Size Increasing

Un programme est NSI si le calcul de $p(x)$ s'effectue en espace $|x| + \alpha$.
Un programme NSI peut être compilé en un programme `C` sans `malloc`.

(Hofmann, 98)

Non Size Increasing

Un programme est NSI si le calcul de $p(x)$ s'effectue en espace $|x| + \alpha$.
Un programme NSI peut être compilé en un programme `C` sans `malloc`.

(Hofmann, 98)

$$\boxed{\text{QI somme multilinéaire} \equiv \text{NSI}}$$

(Amadio, 04)

Non Size Increasing

Un programme est NSI si le calcul de $p(x)$ s'effectue en espace $|x| + \alpha$.
Un programme NSI peut être compilé en un programme `C` sans `malloc`.

(Hofmann, 98)

$$\boxed{\text{QI somme multilinéaire} \equiv \text{NSI}}$$

(Amadio, 04)

Conclusion

- Outil performant, qui a fait ses preuves.
- Recherche sur l'outil autant qu'avec l'outil :
  - Synthèse de QI sachant que le programme termine par MPO.
  - Sup-interprétations : pour les programmes dont le résultat est plus petit que les entrées (soustrazction, quotient).
- Problèmes des algorithmes "Diviser pour régner" (intensionnalité).