UNIVERSITÉ PARIS-EST

lacl

Caractérisation impérative des algorithmes séquentiels en temps quelconque, primitif récursif ou polynomial

Yoann Marquer

École doctorale MSTIC, ANR TARMAC

9 octobre 2015

Fonction : relation entrée-sortie.

$$(x, y) \mapsto x \times y$$

Fonction : relation entrée-sortie.

$$(x, y) \mapsto x \times y$$

Algorithme : manière de calculer.

$$\begin{array}{rr} & 3\ 4 \\ \times & 1\ 2 \\ \hline & 6\ 8 \\ 3\ 4 & \cdot \\ \hline 4\ 0\ 8 & \end{array}$$

Fonction : relation entrée-sortie.

$$(x, y) \mapsto x \times y$$

Algorithme : manière de calculer.

$$\begin{array}{r} 3\ 4 \\ \times\ 1\ 2 \\ \hline 6\ 8 \\ 3\ 4\ \cdot \\ \hline 4\ 0\ 8 \end{array}$$

$$(10a + b) \times (10c + d)$$
$$= 100ac + 10(ad + bc) + bd$$
$$\rightarrow 4 \text{ multiplications}$$

Fonction : relation entrée-sortie.

$$(x, y) \mapsto x \times y$$

Algorithme scolaire :

$$\begin{array}{r} 3\ 4 \\ \times\ 1\ 2 \\ \hline 6\ 8 \\ 3\ 4\ \cdot \\ \hline 4\ 0\ 8 \end{array}$$

$$(10a + b) \times (10c + d)$$
$$= 100ac + 10(ad + bc) + bd$$
$$\rightarrow 4 \text{ multiplications}$$

Algorithme de Karatsuba (1960) :

$$(10a + b) \times (10c + d)$$
$$= 100ac + 10(ac + bd - (a - b)(c - d)) + bd$$
$$\rightarrow 3 \text{ multiplications}$$

Plan

Que pouvons-nous calculer effectivement ?

(film : machine de Turing)

L'intuition de « méthode effective de calcul » est formalisée par :

L'intuition de « méthode effective de calcul » est formalisée par :

- les fonctions récursives de Gödel et Herbrand (1934)

L'intuition de « méthode effective de calcul » est formalisée par :

- les fonctions récursives de Gödel et Herbrand (1934)
- le lambda-calcul de Church et Kleene (1936)

L'intuition de « méthode effective de calcul » est formalisée par :

- les fonctions récursives de Gödel et Herbrand (1934)
- le lambda-calcul de Church et Kleene (1936)
- les machines de Turing (1936)

L'intuition de « méthode effective de calcul » est formalisée par :

- les fonctions récursives de Gödel et Herbrand (1934)
- le lambda-calcul de Church et Kleene (1936)
- les machines de Turing (1936)
- etc.

L'intuition de « méthode effective de calcul » est formalisée par :

- les fonctions récursives de Gödel et Herbrand (1934)
- le lambda-calcul de Church et Kleene (1936)
- les machines de Turing (1936)
- etc.

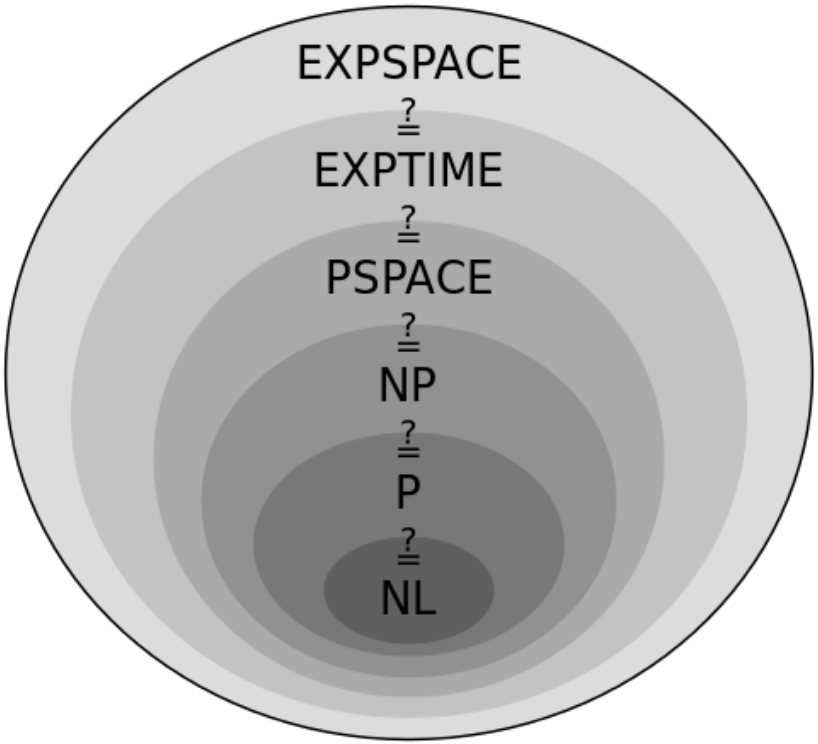
Ces modèles se simulent les uns dans les autres ! ☺

Thèse (Church, 1943)

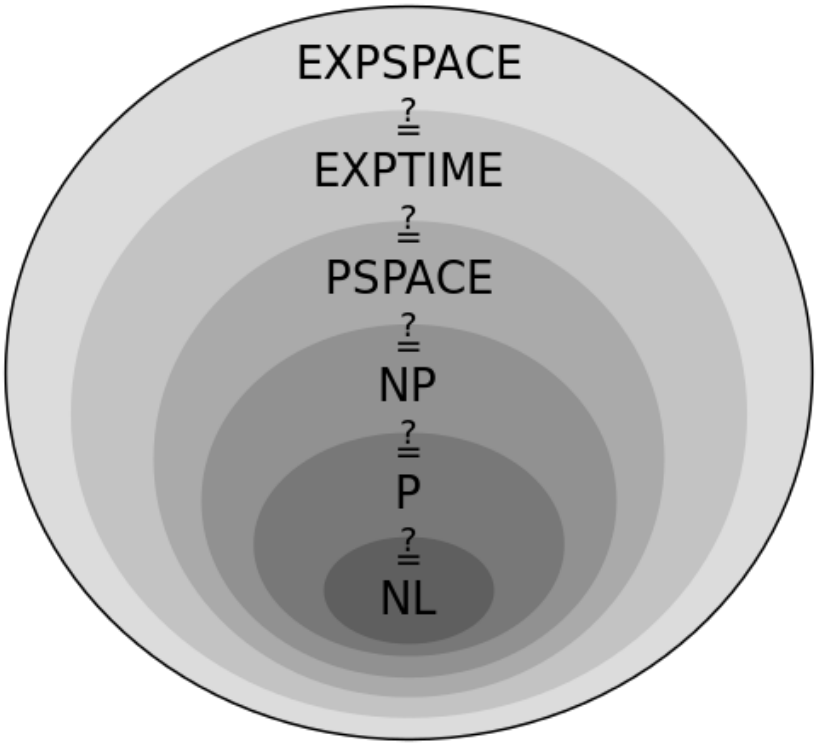
effectivement calculable = calculable par une machine de Turing

Raffiner la notion ?

Raffiner la notion ?



Raffiner la notion ?



- Automates (1943)
- Algorithmes de Kolmogorov (1953)
- Algorithmes de Markov (1960)
- Grigoriev (1976)
- Machines de Gandy (1980)
- etc.

Problème ( )

machine de Turing :

Problème ( )

Reconnaissance de palindrome par une machine de Turing :

- *$O(n)$ à partir de deux rubans*

Problème (équivalence algorithmique)

Reconnaissance de palindrome par une machine de Turing :

- $O(n)$ *à partir de deux rubans*
- $O(n^2/log(n))$ *étapes avec un ruban (2014)*

Problème (équivalence algorithmique)

Reconnaissance de palindrome par une machine de Turing :

- $O(n)$ *à partir de deux rubans*
- $O(n^2/log(n))$ *étapes avec un ruban (2014)*

Problème ( )

La récursion primitive

Problème (équivalence algorithmique)

Reconnaissance de palindrome par une machine de Turing :

- $O(n)$ *à partir de deux rubans*
- $O(n^2/log(n))$ *étapes avec un ruban (2014)*

Problème (complétude algorithmique)

La récursion primitive ne permet pas de calculer :

- *min en* $O(min(m, n))$ *étapes (Colson, 1991)*

Problème (équivalence algorithmique)

Reconnaissance de palindrome par une machine de Turing :

- $O(n)$ *à partir de deux rubans*
- $O(n^2/log(n))$ *étapes avec un ruban (2014)*

Problème (complétude algorithmique)

La récursion primitive ne permet pas de calculer :

- *min en* $O(min(m, n))$ *étapes (Colson, 1991)*
- *pgcd en* $O(log(m) + log(n))$ *étapes (Moschovakis, 2003)*

Formaliser les algorithmes ?

Formaliser les algorithmes ?

- Abstract State Machines (Gurevich, 2000)

Formaliser les algorithmes ?

- Abstract State Machines (Gurevich, 2000)
- Système d'équations récursives (Moschovakis, 2003)

Formaliser les algorithmes ?

- Abstract State Machines (Gurevich, 2000)
- Système d'équations récursives (Moschovakis, 2003)
- Théorie de Galois des algorithmes (Yanofsky, 2011)

Formaliser les algorithmes ?

- Abstract State Machines (Gurevich, 2000)
- Système d'équations récursives (Moschovakis, 2003)
- Théorie de Galois des algorithmes (Yanofsky, 2011)

A priori pas d'équivalence... ☹

Plan

Une présentation axiomatique.

Une présentation axiomatique.

Thèse (Gurevich, 2000)

Les algorithmes séquentiels (`Algo`) sont définis par trois postulats :

- ***Temps séquentiel*** *: exécution étape par étape*
- ***États abstraits*** *: structures de données représentables*
- ***Exploration bornée*** *: quantité bornée de lecture/écriture*

Postulat (Temps séquentiel)

Un algorithme (séquentiel) est la donnée :

Postulat (Temps séquentiel)

Un algorithme (séquentiel) est la donnée :

- *d'un ensemble d'états $S(A)$*

Postulat (Temps séquentiel)

Un algorithme (séquentiel) est la donnée :

- *d'un ensemble d'états* $S(A)$
- *d'un ensemble d'états initiaux* $I(A) \subseteq S(A)$

Postulat (Temps séquentiel)

Un algorithme (séquentiel) est la donnée :

- *d'un ensemble d'états* $S(A)$
- *d'un ensemble d'états initiaux* $I(A) \subseteq S(A)$
- *d'une fonction de transition* $\tau_A : S(A) \to S(A)$

Postulat (Temps séquentiel)

Un algorithme (séquentiel) est la donnée :

- *d'un ensemble d'états* $S(A)$
- *d'un ensemble d'états initiaux* $I(A) \subseteq S(A)$
- *d'une fonction de transition* $\tau_A : S(A) \to S(A)$

Une **exécution** de A est une suite $\vec{X} = X_0, X_1, X_2, \ldots$ telle que :

- X_0 est un état initial
- pour tout $i \in \mathbb{N}$: $X_{t+1} = \tau_A(X_t)$

Postulat (Temps séquentiel)

Un algorithme (séquentiel) est la donnée :

- *d'un ensemble d'états* $S(A)$
- *d'un ensemble d'états initiaux* $I(A) \subseteq S(A)$
- *d'une fonction de transition* $\tau_A : S(A) \to S(A)$

Une **exécution** de A est une suite $\vec{X} = X_0, X_1, X_2, \ldots$ telle que :

- X_0 est un état initial
- pour tout $i \in \mathbb{N}$: $X_{t+1} = \tau_A(X_t)$

Elle est **terminale** s'il existe X_t telle que $X_{t+1} = X_t$.

Postulat (Temps séquentiel)

Un algorithme (séquentiel) est la donnée :

- *d'un ensemble d'états* $S(A)$
- *d'un ensemble d'états initiaux* $I(A) \subseteq S(A)$
- *d'une fonction de transition* $\tau_A : S(A) \to S(A)$

Une **exécution** de A est une suite $\vec{X} = X_0, X_1, X_2, \ldots$ telle que :

- X_0 est un état initial
- pour tout $i \in \mathbb{N}$: $X_{t+1} = \tau_A(X_t)$

Elle est **terminale** s'il existe X_t telle que $X_{t+1} = X_t$.

$$time(A, X_0) =_{def} \begin{cases} min\{t \in \mathbb{N} \mid X_{t+1} = X_t\} \\ \infty \end{cases}$$

Postulat (États abstraits)

- *Les états de A sont des structures de même langage $\mathcal{L}(A)$*
- *$S(A)$ et $I(A)$ sont clos par isomorphismes*
- *τ_A conserve les univers des états et commute avec les isomorphismes*

Postulat (États abstraits)

- *Les états de A sont des structures de même langage $\mathcal{L}(A)$*
- *$S(A)$ et $I(A)$ sont clos par isomorphismes*
- *τ_A conserve les univers des états et commute avec les isomorphismes*

où une structure X est la donnée :

Postulat (États abstraits)

- *Les états de A sont des structures de même langage $\mathcal{L}(A)$*
- *$S(A)$ et $I(A)$ sont clos par isomorphismes*
- *τ_A conserve les univers des états et commute avec les isomorphismes*

où une structure X est la donnée :

- d'un langage : ensemble de symboles $\mathcal{L}(X)$ (syntaxe)

Postulat (États abstraits)

- *Les états de A sont des structures de même langage $\mathcal{L}(A)$*
- *$S(A)$ et $I(A)$ sont clos par isomorphismes*
- *τ_A conserve les univers des états et commute avec les isomorphismes*

où une structure X est la donnée :

- d'un langage : ensemble de symboles $\mathcal{L}(X)$ (syntaxe)
- d'un univers : ensemble d'éléments $\mathcal{U}(X)$ (sémantique)

Postulat (États abstraits)

- *Les états de A sont des structures de même langage $\mathcal{L}(A)$*
- *$S(A)$ et $I(A)$ sont clos par isomorphismes*
- *τ_A conserve les univers des états et commute avec les isomorphismes*

où une structure X est la donnée :

- d'un langage : ensemble de symboles $\mathcal{L}(X)$ (syntaxe)
- d'un univers : ensemble d'éléments $\mathcal{U}(X)$ (sémantique)
- pour chaque symbole d'une interprétation dans l'univers

Postulat (Exploration bornée)

Il existe un ensemble fini T de termes du langage de A tel que si deux états X et Y coïncident sur T alors $\Delta(A, X) = \Delta(A, Y)$.

Postulat (Exploration bornée)

Il existe un ensemble fini T de termes du langage de A tel que si deux états X et Y coïncident sur T alors $\Delta(A, X) = \Delta(A, Y)$.

Nous distinguerons dans $\mathcal{L}(A)$:

- $Dyn(A)$: les symboles **dynamiques**
- $Stat(A)$: les symboles **statiques**

Postulat (Exploration bornée)

Il existe un ensemble fini T de termes du langage de A tel que si deux états X et Y coïncident sur T alors $\Delta(A, X) = \Delta(A, Y)$.

Nous distinguerons dans $\mathcal{L}(A)$:

- $Dyn(A)$: les symboles **dynamiques**
- $Stat(A)$: les symboles **statiques**

Mise à jour de f en $(t_1, \ldots, t_\alpha)$ par la valeur t_0 :

$$f(t_1, \ldots, t_\alpha) := t_0$$

Postulat (Exploration bornée)

Il existe un ensemble fini T de termes du langage de A tel que si deux états X et Y coïncident sur T alors $\Delta(A, X) = \Delta(A, Y)$.

Nous distinguerons dans $\mathcal{L}(A)$:

- $Dyn(A)$: les symboles **dynamiques**
- $Stat(A)$: les symboles **statiques**

Mise à jour de f en $(t_1, \ldots, t_\alpha)$ par la valeur t_0 :

$$f(t_1, \ldots, t_\alpha) := t_0$$

Ensemble de mises à jour Δ

Postulat (Exploration bornée)

Il existe un ensemble fini T de termes du langage de A tel que si deux états X et Y coïncident sur T alors $\Delta(A, X) = \Delta(A, Y)$.

Nous distinguerons dans $\mathcal{L}(A)$:

- $Dyn(A)$: les symboles **dynamiques**
- $Stat(A)$: les symboles **statiques**

Mise à jour de f en $(t_1, \ldots, t_\alpha)$ par la valeur t_0 :

$$f(t_1, \ldots, t_\alpha) := t_0$$

Ensemble de mises à jour Δ inconsistant si :

$$\left.\begin{array}{l} f(t_1, \ldots, t_\alpha) := t_0 \\ f(t_1, \ldots, t_\alpha) := \theta_0 \end{array}\right\} \in \Delta \text{ avec } t_0 \neq \theta_0$$

Une présentation opérationnelle.

Une présentation opérationnelle.

Définition (Programmes d'ASM)

$$\begin{aligned}\Pi =_{def}\ & f(t_1,\ldots,t_\alpha) := t_0 \\ & \mid \texttt{if } F \texttt{ then } \Pi_1 \texttt{ else } \Pi_2 \texttt{ endif} \\ & \mid \texttt{par } \Pi_1 \| \ldots \| \Pi_n \texttt{ endpar}\end{aligned}$$

⚠ Le `par` est simultané !

Une présentation opérationnelle.

Définition (Programmes d'ASM)

$$\begin{aligned}\Pi =_{def}\ & f(t_1,\ldots,t_\alpha) := t_0 \\ & \mid \texttt{if } F \texttt{ then } \Pi_1 \texttt{ else } \Pi_2 \texttt{ endif} \\ & \mid \texttt{par } \Pi_1 \| \ldots \| \Pi_n \texttt{ endpar}\end{aligned}$$

⚠ Le `par` est simultané !

ASM est un modèle de calcul pour les algorithmes séquentiels :

Théorème (Gurevich, 2000)

$$\texttt{Algo} = \texttt{ASM}$$

Un programme d'ASM pour le minimum :

$$\texttt{if } \neg(r = m \vee r = n) \texttt{ then } r := r + 1 \texttt{ endif}$$

Un programme d'ASM pour le minimum :

$$\texttt{if } \neg(r = m \vee r = n) \texttt{ then } r := r + 1 \texttt{ endif}$$

Entrées : $m = 2, n = 3$.

Un programme d'ASM pour le minimum :

$$\texttt{if } \neg(r = m \vee r = n) \texttt{ then } r := r + 1 \texttt{ endif}$$

Entrées : $m = 2, n = 3$.

- $r = 0$

Un programme d'ASM pour le minimum :

$$\texttt{if } \neg(r = m \vee r = n) \texttt{ then } r := r + 1 \texttt{ endif}$$

Entrées : $m = 2, n = 3$.

- $r = 0$

  $r = 2$ ou $r = 3$?

Un programme d'ASM pour le minimum :

$$\texttt{if } \neg(r = m \vee r = n) \texttt{ then } r := r + 1 \texttt{ endif}$$

Entrées : $m = 2, n = 3$.

- $r = 0$
  $r = 2$ ou $r = 3$? non ! $r := 0 + 1$
- $r = 1$

Un programme d'ASM pour le minimum :

$$\texttt{if } \neg(r = m \vee r = n) \texttt{ then } r := r + 1 \texttt{ endif}$$

Entrées : $m = 2, n = 3$.

- $r = 0$
  $r = 2$ ou $r = 3$? non ! $r := 0 + 1$
- $r = 1$
  $r = 2$ ou $r = 3$?

Un programme d'ASM pour le minimum :

$$\texttt{if } \neg(r = m \vee r = n) \texttt{ then } r := r + 1 \texttt{ endif}$$

Entrées : $m = 2, n = 3$.

- $r = 0$
  $r = 2$ ou $r = 3$? non ! $r := 0 + 1$
- $r = 1$
  $r = 2$ ou $r = 3$? non ! $r := 1 + 1$
- $r = 2$

Un programme d'ASM pour le minimum :

$$\texttt{if } \neg(r = m \vee r = n) \texttt{ then } r := r + 1 \texttt{ endif}$$

Entrées : $m = 2, n = 3$.

- $r = 0$
  $r = 2$ ou $r = 3$? non ! $r := 0 + 1$
- $r = 1$
  $r = 2$ ou $r = 3$? non ! $r := 1 + 1$
- $r = 2$
  $r = 2$ ou $r = 3$?

Un programme d'ASM pour le minimum :

$$\texttt{if } \neg(r = m \vee r = n) \texttt{ then } r := r + 1 \texttt{ endif}$$

Entrées : $m = 2, n = 3$.

- $r = 0$
  $r = 2$ ou $r = 3$? non ! $r := 0 + 1$
- $r = 1$
  $r = 2$ ou $r = 3$? non ! $r := 1 + 1$
- $r = 2$
  $r = 2$ ou $r = 3$? oui !

Sortie : $min(2, 3) = 2$

Plan

Définition (Programmes impératifs)

$$
\begin{aligned}
c =_{def}\ & f(t_1, \ldots, t_\alpha) := t_0 \\
\mid\ & \texttt{if } F \texttt{ } \{P_1\} \texttt{ else } \{P_2\} \\
\mid\ & \texttt{while } F \texttt{ } \{P\} \\
\mid\ & \texttt{loop } n \texttt{ except } F \texttt{ } \{P\} \\
P =_{def}\ & \texttt{end} \\
\mid\ & c; P
\end{aligned}
$$

où n est une variable statique dans P.

Définition (Programmes impératifs)

$$
\begin{aligned}
c =_{def}\ & f(t_1, \ldots, t_\alpha) := t_0 \\
\mid\ & \texttt{if } F \texttt{ } \{P_1\} \texttt{ else } \{P_2\} \\
\mid\ & \texttt{while } F \texttt{ } \{P\} \\
\mid\ & \texttt{loop } n \texttt{ except } F \texttt{ } \{P\} \\
P =_{def}\ & \texttt{end} \\
\mid\ & c; P
\end{aligned}
$$

où n est une variable statique dans P.

- `While` : mises à jour, `if` et boucles `while`

Définition (Programmes impératifs)

$$
\begin{aligned}
c =_{def}\ & f(t_1, \ldots, t_\alpha) := t_0 \\
\mid\ & \texttt{if } F \texttt{ } \{P_1\} \texttt{ else } \{P_2\} \\
\mid\ & \texttt{while } F \texttt{ } \{P\} \\
\mid\ & \texttt{loop } n \texttt{ except } F \texttt{ } \{P\} \\
P =_{def}\ & \texttt{end} \\
\mid\ & c; P
\end{aligned}
$$

où n est une variable statique dans P.

- `While` : mises à jour, `if` et boucles `while`
- `LoopC` : mises à jour, `if` et boucles `loop` conditionnées

Un programme de LoopC pour le minimum :

$$r := 0; \texttt{loop } n \texttt{ except } r = m \{ r := r + 1; \}; \texttt{end}$$

Un programme de LoopC pour le minimum :

$$r := 0; \texttt{loop } n \texttt{ except } r = m \{r := r + 1; \}; \texttt{end}$$

Minimum entre $m = 2$ et $n = 3$?

$$r := 0; \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1; \}; \texttt{end}$$

Un programme de LoopC pour le minimum :

$$r := 0; \texttt{loop}\ n\ \texttt{except}\ r = m\ \{r := r + 1;\}; \texttt{end}$$

Minimum entre $m = 2$ et $n = 3$?

$$r := 0; \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\}; \texttt{end} \star [i = 0, r = ?]$$

Un programme de LoopC pour le minimum :

$$r := 0; \texttt{loop}\ n\ \texttt{except}\ r = m\ \{r := r + 1;\}; \texttt{end}$$

Minimum entre $m = 2$ et $n = 3$?

$$\begin{aligned} & r := 0; \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\}; \texttt{end} \star [i = 0, r = ?] \\ \succ\ & \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\}; \texttt{end} \star [i = 0, r = 0] \end{aligned}$$

Un programme de LoopC pour le minimum :

$$r := 0; \texttt{loop } n \texttt{ except } r = m \{r := r + 1;\}; \texttt{end}$$

Minimum entre $m = 2$ et $n = 3$?

$$\begin{aligned} & r := 0; \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1;\}; \texttt{end} \star [i = 0, r = ?] \\ \succ \quad & \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1;\}; \texttt{end} \star [i = 0, r = 0] \\ \succ \quad & r := r + 1; \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1;\}; \texttt{end} \star [i = 1, r = 0] \end{aligned}$$

Un programme de LoopC pour le minimum :

$$r := 0; \texttt{loop } n \texttt{ except } r = m \{r := r + 1;\}; \texttt{end}$$

Minimum entre $m = 2$ et $n = 3$?

$$\begin{aligned}
& r := 0; \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1;\}; \texttt{end} \star [i = 0, r = ?] \\
\succ \quad & \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1;\}; \texttt{end} \star [i = 0, r = 0] \\
\succ \quad & r := r + 1; \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1;\}; \texttt{end} \star [i = 1, r = 0] \\
\succ \quad & \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1;\}; \texttt{end} \star [i = 1, r = 1]
\end{aligned}$$

Un programme de LoopC pour le minimum :

$$r := 0; \texttt{loop}\ n\ \texttt{except}\ r = m\ \{r := r + 1;\};\texttt{end}$$

Minimum entre $m = 2$ et $n = 3$?

$$\begin{aligned}
& r := 0; \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\};\texttt{end} \star [i = 0, r = ?] \\
\succ\ & \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\};\texttt{end} \star [i = 0, r = 0] \\
\succ\ & r := r + 1; \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\};\texttt{end} \star [i = 1, r = 0] \\
\succ\ & \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\};\texttt{end} \star [i = 1, r = 1] \\
\succ\ & r := r + 1; \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\};\texttt{end} \star [i = 2, r = 1]
\end{aligned}$$

Un programme de LoopC pour le minimum :

$$r := 0; \texttt{loop } n \texttt{ except } r = m \{r := r + 1;\}; \texttt{end}$$

Minimum entre $m = 2$ et $n = 3$?

$$\begin{aligned}
& r := 0; \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1;\}; \texttt{end} \star [i = 0, r = ?] \\
\succ \quad & \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1;\}; \texttt{end} \star [i = 0, r = 0] \\
\succ \quad & r := r + 1; \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1;\}; \texttt{end} \star [i = 1, r = 0] \\
\succ \quad & \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1;\}; \texttt{end} \star [i = 1, r = 1] \\
\succ \quad & r := r + 1; \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1;\}; \texttt{end} \star [i = 2, r = 1] \\
\succ \quad & \texttt{loop } 3 \texttt{ except } r = 2 \{r := r + 1;\}; \texttt{end} \star [i = 2, r = 2]
\end{aligned}$$

Un programme de LoopC pour le minimum :

$$r := 0; \texttt{loop}\ n\ \texttt{except}\ r = m\ \{r := r + 1;\};\texttt{end}$$

Minimum entre $m = 2$ et $n = 3$?

$$\begin{aligned}
& r := 0; \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\};\texttt{end} \star [i = 0, r = ?] \\
\succ\ & \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\};\texttt{end} \star [i = 0, r = 0] \\
\succ\ & r := r + 1; \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\};\texttt{end} \star [i = 1, r = 0] \\
\succ\ & \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\};\texttt{end} \star [i = 1, r = 1] \\
\succ\ & r := r + 1; \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\};\texttt{end} \star [i = 2, r = 1] \\
\succ\ & \texttt{loop}\ 3\ \texttt{except}\ r = 2\ \{r := r + 1;\};\texttt{end} \star [i = 2, r = 2] \\
\succ\ & \texttt{end} \star [i = 0, r = 2]
\end{aligned}$$

Résultat : $r = 2$

Si $P \star X \succ_t \texttt{end} \star Y$, alors P **termine** sur X.

Si $P \star X \succ_t \texttt{end} \star Y$, alors P **termine** sur X.
t est noté $time(P, X)$, et Y est noté $P(X)$, d'où :

$$P \star X \succ_{time(P,X)} \texttt{end} \star P(X)$$

Si $P \star X \succ_t \texttt{end} \star Y$, alors P **termine** sur X.
t est noté $time(P, X)$, et Y est noté $P(X)$, d'où :

$$P \star X \succ_{time(P,X)} \texttt{end} \star P(X)$$

Un programme sera dit terminal s'il termine sur tous ses états.

Si $P \star X \succ_t \texttt{end} \star Y$, alors P **termine** sur X.
t est noté $time(P, X)$, et Y est noté $P(X)$, d'où :

$$P \star X \succ_{time(P,X)} \texttt{end} \star P(X)$$

Un programme sera dit terminal s'il termine sur tous ses états.
Exemple de programme non terminal :

$$\texttt{while } true \ \{\}$$

Si $P \star X \succ_t \texttt{end} \star Y$, alors P **termine** sur X.
t est noté $time(P, X)$, et Y est noté $P(X)$, d'où :

$$P \star X \succ_{time(P,X)} \texttt{end} \star P(X)$$

Un programme sera dit terminal s'il termine sur tous ses états.
Exemple de programme non terminal :

$$\texttt{while } true \ \{\}$$

Proposition (Terminaison)

Les programmes de LoopC *sont terminaux.*

Plan

Définition (Complexité en temps)

$A \in \texttt{Algo}$ est en temps $\mathcal{C}$
s'il existe $\varphi_A \in \mathcal{C}$ tel que pour tout état initial X_0 :

$$time(A, X_0) \leq \varphi_A(|X_0|)$$

où $|X|$ est la taille de l'état X.

Définition (Complexité en temps)

$A \in \texttt{Algo}$ est en temps $\mathcal{C}$
s'il existe $\varphi_A \in \mathcal{C}$ tel que pour tout état initial X_0 :

$$time(A, X_0) \leq \varphi_A(|X_0|)$$

où $|X|$ est la taille de l'état X.

- `Algo` : tous les algorithmes
  $\rightarrow$ effectivement calculables

Définition (Complexité en temps)

$A \in \texttt{Algo}$ est en temps $\mathcal{C}$
s'il existe $\varphi_A \in \mathcal{C}$ tel que pour tout état initial X_0 :

$$time(A, X_0) \leq \varphi_A(|X_0|)$$

où $|X|$ est la taille de l'état X.

- $\texttt{Algo}$: tous les algorithmes
  $\rightarrow$ effectivement calculables
- $\texttt{Algo}_{\mathcal{PR}}$: les algorithmes en temps primitif récursif
  $\rightarrow$ usuels, terminaux

Définition (Complexité en temps)

$A \in \texttt{Algo}$ est en temps $\mathcal{C}$
s'il existe $\varphi_A \in \mathcal{C}$ tel que pour tout état initial X_0 :

$$time(A, X_0) \leq \varphi_A(|X_0|)$$

où $|X|$ est la taille de l'état X.

- $\texttt{Algo}$: tous les algorithmes
  $\rightarrow$ effectivement calculables
- $\texttt{Algo}_{\mathcal{PR}}$: les algorithmes en temps primitif récursif
  $\rightarrow$ usuels, terminaux
- $\texttt{Algo}_{\mathcal{P}ol}$: les algorithmes en temps polynomial
  $\rightarrow$ en coût raisonnable

Des programmes en temps primitif récursif ?

Des programmes en temps primitif récursif ?

$n := ack(n, n);$
$\texttt{loop}\ n\ \{\}$

Des programmes en temps primitif récursif ?

$n := ack(n, n);$
`loop` n $\{\}$

Définition (Classe des opérations)

Une opération f est en espace $\mathcal{C}$ s'il existe $\varphi_f \in \mathcal{C}$ telle que pour tout état X :

$$|\overline{f}^X(\vec{a})| \leq \varphi_f(|\vec{a}|)$$

Des programmes en temps primitif récursif ?

$n := ack(n, n);$
$\texttt{loop}\ n\ \{\}$

Définition (Classe des opérations)

Une opération f est en espace $\mathcal{C}$ s'il existe $\varphi_f \in \mathcal{C}$ telle que pour tout état X :

$$|\overline{f}^X(\vec{a})| \leq \varphi_f(|\vec{a}|)$$

Proposition (Temps primitif récursif)

Pour tout $P \in \texttt{LoopC}$, si les opérations sont en espace $\mathcal{PR}$ alors il existe $\varphi_P \in \mathcal{PR}$ telle que pour tout X :

$$time(P, X) \leq \varphi_P(|X|)$$

Des programmes en temps polynomial ?

Des programmes en temps polynomial ?

Définition (Restriction syntaxique)

$\texttt{LoopC}_{\texttt{stat}} =_{def} \{P \in \texttt{LoopC} \mid \forall \texttt{loop}\ n\ \{Q\} \in P, n \in Stat(P)\}$

Des programmes en temps polynomial ?

Définition (Restriction syntaxique)

$\texttt{LoopC}_{\texttt{stat}} =_{def} \{P \in \texttt{LoopC} \mid \forall \texttt{loop}\ n\ \{Q\} \in P, n \in Stat(P)\}$

Proposition (Temps polynomial)

Pour tout $P \in \texttt{LoopC}_{\texttt{stat}}$ *il existe* $\varphi_P \in \mathcal{P}ol$ *telle que pour tout* X *:*

$$time(P, X) \leq \varphi_P(|X|)$$

Des programmes en temps polynomial ?

Définition (Restriction syntaxique)

$\texttt{LoopC}_{\texttt{stat}} =_{def} \{P \in \texttt{LoopC} \mid \forall \texttt{loop}\ n\ \{Q\} \in P, n \in Stat(P)\}$

Proposition (Temps polynomial)

Pour tout $P \in \texttt{LoopC}_{\texttt{stat}}$ *il existe* $\varphi_P \in \mathcal{P}ol$ *telle que pour tout* X *:*

$$time(P, X) \leq \varphi_P(|X|)$$

De plus : $deg(\varphi_P) = depth(P)$

Plan


Être, c'est être perçu ou percevoir. — (George Berkeley)

Variables temporaires

Variables temporaires

Exemple (Simulation d'un `loop` par un `while`)

$$\texttt{loop}\ n\ \{P\}$$

$$i := 0; \texttt{while}\ i \neq n\ \{P; i := i + 1\}$$

Variables temporaires

Exemple (Simulation d'un `loop` par un `while`)

$$\texttt{loop}\ n\ \{P\}$$

$$i := 0; \texttt{while}\ i \neq n\ \{P; i := i + 1\}$$

Exemple (Echange de deux valeurs)

$$x \leftrightarrow y$$

$$v := x; x := y; y := v$$

Dilatation temporelle

Dilatation temporelle

$$\frac{M_3 \qquad M_1}{X_0 \quad = \quad Y_0}$$

Dilatation temporelle

$$\begin{array}{cc} M_3 & M_1 \\ \hline X_0 & = & Y_0 \\ X_1 \end{array}$$

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
X_3	$=$	Y_1

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
X_3	$=$	Y_1
X_4		

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
X_3	$=$	Y_1
X_4		
X_5		

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
X_3	$=$	Y_1
X_4		
X_5		
X_6	$=$	Y_2

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
X_3	$=$	Y_1
X_4		
X_5		
X_6	$=$	Y_2
X_7		

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
X_3	$=$	Y_1
X_4		
X_5		
X_6	$=$	Y_2
X_7		
X_8		

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
X_3	$=$	Y_1
X_4		
X_5		
X_6	$=$	Y_2
X_7		
X_8		
$\vdots$		$\vdots$

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
X_3	$=$	Y_1
X_4		
X_5		
X_6	$=$	Y_2
X_7		
X_8		
$\vdots$		$\vdots$

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
$\vdots$		$\vdots$

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
X_3	$=$	Y_1
X_4		
X_5		
X_6	$=$	Y_2
X_7		
X_8		
$\vdots$		$\vdots$

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
$\vdots$		$\vdots$
X_{3t+0}	$=$	Y_t

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
X_3	$=$	Y_1
X_4		
X_5		
X_6	$=$	Y_2
X_7		
X_8		
$\vdots$		$\vdots$

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
$\vdots$		$\vdots$
X_{3t+0}	$=$	Y_t
X_{3t+1}		

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
X_3	$=$	Y_1
X_4		
X_5		
X_6	$=$	Y_2
X_7		
X_8		
$\vdots$		$\vdots$

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
$\vdots$		$\vdots$
X_{3t+0}	$=$	Y_t
X_{3t+1}		
X_{3t+2}		

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
X_3	$=$	Y_1
X_4		
X_5		
X_6	$=$	Y_2
X_7		
X_8		
$\vdots$		$\vdots$

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
$\vdots$		$\vdots$
X_{3t+0}	$=$	Y_t
X_{3t+1}		
X_{3t+2}		
X_{3t+0}	$=$	Y_t

Dilatation temporelle

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
X_3	$=$	Y_1
X_4		
X_5		
X_6	$=$	Y_2
X_7		
X_8		
$\vdots$		$\vdots$

M_3		M_1
X_0	$=$	Y_0
X_1		
X_2		
$\vdots$		$\vdots$
X_{3t+0}	$=$	Y_t
X_{3t+1}		
X_{3t+2}		
X_{3t+0}	$=$	Y_t
$\vdots$		$\vdots$

Définition (M_1 simule M_2)

Pour tout programme P_2 de M_2
il existe un programme P_1 de M_1 tel que :

Définition (M_1 simule M_2)

Pour tout programme P_2 de M_2
il existe un programme P_1 de M_1 tel que :

- $\mathcal{L}(P_1) \supseteq \mathcal{L}(P_2)$
  et $\mathcal{L}(P_1) \setminus \mathcal{L}(P_2)$ est un ensemble fini de variables

Définition (M_1 simule M_2)

Pour tout programme P_2 de M_2
il existe un programme P_1 de M_1 tel que :

- $\mathcal{L}(P_1) \supseteq \mathcal{L}(P_2)$
  et $\mathcal{L}(P_1) \setminus \mathcal{L}(P_2)$ est un ensemble fini de variables

et il existe $d \in \mathbb{N}^\star$ et $e \in \mathbb{N}$ dépendants seulement de P_2 tels que pour toute exécution $\vec{Y}$ de P_2 il existe une exécution $\vec{X}$ de P_1 tq :

Définition (M_1 simule M_2)

Pour tout programme P_2 de M_2
il existe un programme P_1 de M_1 tel que :

- $\mathcal{L}(P_1) \supseteq \mathcal{L}(P_2)$
  et $\mathcal{L}(P_1) \setminus \mathcal{L}(P_2)$ est un ensemble fini de variables

et il existe $d \in \mathbb{N}^\star$ et $e \in \mathbb{N}$ dépendants seulement de P_2 tels que pour toute exécution $\vec{Y}$ de P_2 il existe une exécution $\vec{X}$ de P_1 tq :

- *pour tout $i \in \mathbb{N}$: $X_{d\times i}|_{\mathcal{L}(P_2)} = Y_i$*

Définition (M_1 simule M_2)

Pour tout programme P_2 *de* M_2
il existe un programme P_1 *de* M_1 *tel que :*

- $\mathcal{L}(P_1) \supseteq \mathcal{L}(P_2)$
  et $\mathcal{L}(P_1) \setminus \mathcal{L}(P_2)$ *est un ensemble fini de variables*

et il existe $d \in \mathbb{N}^\star$ *et* $e \in \mathbb{N}$ *dépendants seulement de* P_2 *tels que pour toute exécution* $\vec{Y}$ *de* P_2 *il existe une exécution* $\vec{X}$ *de* P_1 *tq :*

- *pour tout* $i \in \mathbb{N}$: $X_{d\times i}|_{\mathcal{L}(P_2)} = Y_i$
- $time(P_1, X_0) = d \times time(P_2, Y_0) + e$

Définition (M_1 simule M_2)

Pour tout programme P_2 de M_2
il existe un programme P_1 de M_1 tel que :

- $\mathcal{L}(P_1) \supseteq \mathcal{L}(P_2)$
  et $\mathcal{L}(P_1) \setminus \mathcal{L}(P_2)$ est un ensemble fini de variables

et il existe $d \in \mathbb{N}^\star$ et $e \in \mathbb{N}$ dépendants seulement de P_2 tels que pour toute exécution $\vec{Y}$ de P_2 il existe une exécution $\vec{X}$ de P_1 tq :

- *pour tout $i \in \mathbb{N}$: $X_{d\times i}|_{\mathcal{L}(P_2)} = Y_i$*
- $time(P_1, X_0) = d \times time(P_2, Y_0) + e$

Si M_1 simule M_2 et M_2 simule M_1 alors ils seront dits **algorithmiquement équivalents**, et notés $M_1 \simeq M_2$.

Caractérisation impérative des algorithmes séquentiels en temps quelconque, primitif récursif ou polynomial

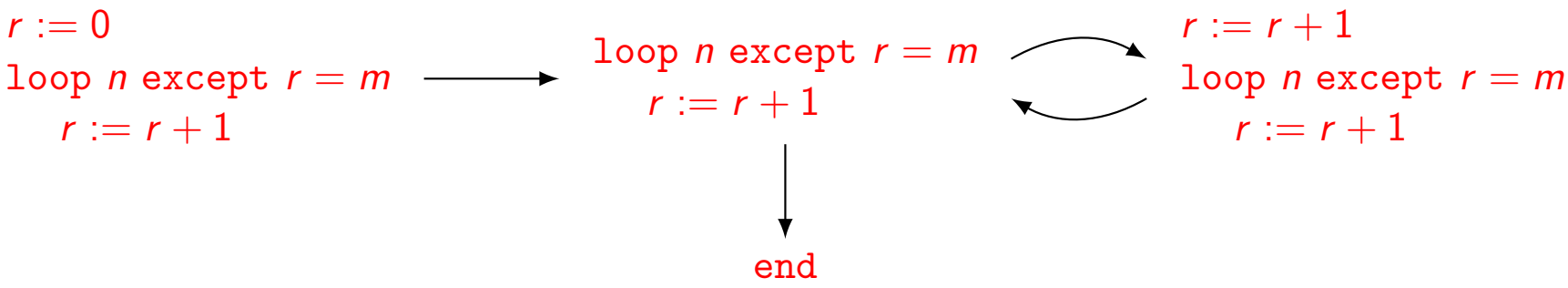
Théorème

- $\texttt{While} \simeq \texttt{ASM}$
- $\texttt{LoopC} \simeq \texttt{ASM}_{\mathcal{PR}}$ *si les opérations sont en espace* $\mathcal{PR}$
- $\texttt{LoopC}_{\texttt{stat}} \simeq \texttt{ASM}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi_{\Pi})$

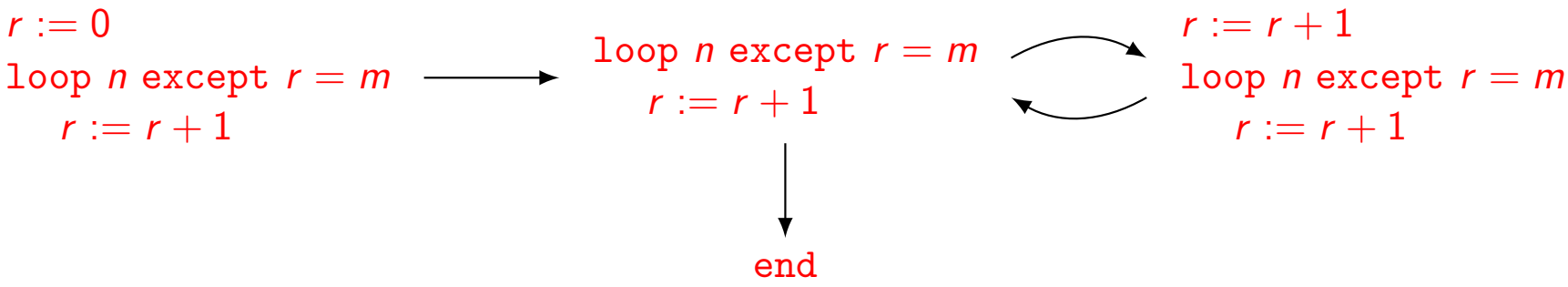
Plan

Ils représentent les exécutions possibles, par exemple pour P_{min} :

Ils représentent les exécutions possibles, par exemple pour P_{min} :



Ils représentent les exécutions possibles, par exemple pour P_{min} :



$$\mathcal{G}(P_{min}) = \{ \quad r := 0; \texttt{loop } n \texttt{ except } r = m \ \{r := r + 1\},$$
$$\texttt{loop } n \texttt{ except } r = m \ \{r := r + 1\},$$
$$r := r + 1; \texttt{loop } n \texttt{ except } r = m \ \{r := r + 1\},$$
$$\texttt{end}$$
$$\}$$

$$\texttt{if } b_{r:=0;\texttt{loop } n \texttt{ except } r=m \{r:=r+1\}} \texttt{ then}$$
$$b_{r:=0;\texttt{loop } n \texttt{ except } r=m \{r:=r+1\}} := \mathit{false}$$
$$r := 0$$
$$b_{\texttt{loop } n \texttt{ except } r=m \{r:=r+1\}} := \mathit{true}$$

if $b_{r:=0;\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}}$ then
  $b_{r:=0;\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}} := \mathit{false}$
  $r := 0$
  $b_{\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}} := \mathit{true}$
if $b_{\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}}$ then
  $b_{\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}} := \mathit{false}$
  if $\neg(i = n \vee r = m)$ then
    $i := i + 1$
    $b_{r:=r+1;\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}} := \mathit{true}$
  else
    $i := 0$
    $b_{\texttt{end}} := \mathit{true}$

if $b_{r:=0;\texttt{loop } n \texttt{ except } r=m \{r:=r+1\}}$ then
  $b_{r:=0;\texttt{loop } n \texttt{ except } r=m \{r:=r+1\}} := \mathit{false}$
  $r := 0$
  $b_{\texttt{loop } n \texttt{ except } r=m \{r:=r+1\}} := \mathit{true}$
if $b_{\texttt{loop } n \texttt{ except } r=m \{r:=r+1\}}$ then
  $b_{\texttt{loop } n \texttt{ except } r=m \{r:=r+1\}} := \mathit{false}$
  if $\neg(i = n \vee r = m)$ then
    $i := i + 1$
    $b_{r:=r+1;\texttt{loop } n \texttt{ except } r=m \{r:=r+1\}} := \mathit{true}$
  else
    $i := 0$
    $b_{\texttt{end}} := \mathit{true}$
if $b_{r:=r+1;\texttt{loop } n \texttt{ except } r=m \{r:=r+1\}}$ then
  $b_{r:=r+1;\texttt{loop } n \texttt{ except } r=m \{r:=r+1\}} := \mathit{false}$
  $r := r + 1$
  $b_{\texttt{loop } n \texttt{ except } r=m \{r:=r+1\}} := \mathit{true}$

if $b_{r:=0;\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}}$ then
  $b_{r:=0;\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}} := \mathit{false}$
  $r := 0$
  $b_{\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}} := \mathit{true}$
if $b_{\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}}$ then
  $b_{\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}} := \mathit{false}$
  if $\neg(i = n \vee r = m)$ then
    $i := i + 1$
    $b_{r:=r+1;\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}} := \mathit{true}$
  else
    $i := 0$
    $b_{\texttt{end}} := \mathit{true}$
if $b_{r:=r+1;\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}}$ then
  $b_{r:=r+1;\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}} := \mathit{false}$
  $r := r + 1$
  $b_{\texttt{loop } n \texttt{ except } r=m\ \{r:=r+1\}} := \mathit{true}$
if $b_{\texttt{end}}$ then

P est donc traduit en le programme d'ASM Π_P.

P est donc traduit en le programme d'ASM $\prod_P$.

- **variables temporaires** :
  $card\{b_{P_j} \mid P_j \in \mathcal{G}(P)\} \leq 1 + length(P)$ nouveaux booléens.

P est donc traduit en le programme d'ASM Π_P.

- **variables temporaires** :
  $card\{b_{P_j} \mid P_j \in \mathcal{G}(P)\} \leq 1 + length(P)$ nouveaux booléens.
- **dilatation temporelle** :
  Pour tout $0 \leq i \leq time(P, X)$: $\tau^i_{\Pi_P}(X[b_P]) = \tau^i_P(X)[b_{\tau^i_X(P)}]$
  donc $d = 1$. De plus on montre que $e = 0$.

P est donc traduit en le programme d'ASM Π_P.

- **variables temporaires** :
  $card\{b_{P_j} \mid P_j \in \mathcal{G}(P)\} \leq 1 + length(P)$ nouveaux booléens.
- **dilatation temporelle** :
  Pour tout $0 \leq i \leq time(P, X)$: $\tau^i_{\Pi_P}(X[b_P]) = \tau^i_P(X)[b_{\tau^i_X(P)}]$
  donc $d = 1$. De plus on montre que $e = 0$.

Théorème (Partie 1)

ASM *simule les programmes impératifs.*

P est donc traduit en le programme d'ASM Π_P.

- **variables temporaires** :
  $card\{b_{P_j} \mid P_j \in \mathcal{G}(P)\} \leq 1 + length(P)$ nouveaux booléens.
- **dilatation temporelle** :
  Pour tout $0 \leq i \leq time(P, X)$: $\tau^i_{\Pi_P}(X[b_P]) = \tau^i_P(X)[b_{\tau^i_X(P)}]$
  donc $d = 1$. De plus on montre que $e = 0$.

Théorème (Partie 1)

ASM *simule les programmes impératifs. En particulier :*

- ASM *simule* While

P est donc traduit en le programme d'ASM Π_P.

- **variables temporaires** :
  $card\{b_{P_j} \mid P_j \in \mathcal{G}(P)\} \leq 1 + length(P)$ nouveaux booléens.
- **dilatation temporelle** :
  Pour tout $0 \leq i \leq time(P, X)$: $\tau^i_{\Pi_P}(X[b_P]) = \tau^i_P(X)[b_{\tau^i_X(P)}]$
  donc $d = 1$. De plus on montre que $e = 0$.

Théorème (Partie 1)

ASM *simule les programmes impératifs. En particulier :*

- ASM *simule* While
- $\text{ASM}_{\mathcal{PR}}$ *simule* LoopC *si les opérations sont en espace* $\mathcal{PR}$

P est donc traduit en le programme d'ASM Π_P.

- **variables temporaires** :
  $card\{b_{P_j} \mid P_j \in \mathcal{G}(P)\} \leq 1 + length(P)$ nouveaux booléens.
- **dilatation temporelle** :
  Pour tout $0 \leq i \leq time(P, X)$: $\tau^i_{\Pi_P}(X[b_P]) = \tau^i_P(X)[b_{\tau^i_X(P)}]$
  donc $d = 1$. De plus on montre que $e = 0$.

Théorème (Partie 1)

ASM *simule les programmes impératifs. En particulier :*

- ASM *simule* While
- $\text{ASM}_{\mathcal{PR}}$ *simule* LoopC *si les opérations sont en espace* $\mathcal{PR}$
- $\text{ASM}_{\mathcal{P}ol}$ *simule* $\text{LoopC}_{\text{stat}}$ *avec* $depth(P) = deg(\varphi_\Pi)$

Plan

La traduction naïve :

$$(ft_1 \dots t_k := t_0)^{tr} =_{def} ft_1 \dots t_k := t_0$$
$$(\texttt{if } F \texttt{ then } \Pi_1 \texttt{ else } \Pi_2 \texttt{ endif})^{tr} =_{def} \texttt{if } F \texttt{ then } \{\Pi_1^{tr}\} \texttt{ else } \{\Pi_2^{tr}\}$$
$$(\texttt{par } \Pi_1 \| \dots \| \Pi_n \texttt{ endpar})^{tr} =_{def} \Pi_1^{tr}; \dots; \Pi_n^{tr}$$

La traduction naïve :

$$(ft_1 \dots t_k := t_0)^{tr} =_{def} ft_1 \dots t_k := t_0$$
$$(\texttt{if } F \texttt{ then } \Pi_1 \texttt{ else } \Pi_2 \texttt{ endif})^{tr} =_{def} \texttt{if } F \texttt{ then } \{\Pi_1^{tr}\} \texttt{ else } \{\Pi_2^{tr}\}$$
$$(\texttt{par } \Pi_1 \| \dots \| \Pi_n \texttt{ endpar})^{tr} =_{def} \Pi_1^{tr}; \dots; \Pi_n^{tr}$$

est insuffisante ! Par exemple le programme :

$$\texttt{par } x := y \| y := x \texttt{ endpar}$$

échange les valeurs de x et y

La traduction naïve :

$$(ft_1 \dots t_k := t_0)^{tr} =_{def} ft_1 \dots t_k := t_0$$
$$(\texttt{if } F \texttt{ then } \Pi_1 \texttt{ else } \Pi_2 \texttt{ endif})^{tr} =_{def} \texttt{if } F \texttt{ then } \{\Pi_1^{tr}\} \texttt{ else } \{\Pi_2^{tr}\}$$
$$(\texttt{par } \Pi_1 \| \dots \| \Pi_n \texttt{ endpar})^{tr} =_{def} \Pi_1^{tr}; \dots; \Pi_n^{tr}$$

est insuffisante ! Par exemple le programme :

$$\texttt{par } x := y \| y := x \texttt{ endpar}$$

échange les valeurs de x et y, alors que sa traduction

$$x := y; y := x$$

écrase la valeur de x par la valeur de y.

La traduction naïve :

$$(ft_1 \dots t_k := t_0)^{tr} =_{def} ft_1 \dots t_k := t_0$$
$$(\texttt{if } F \texttt{ then } \Pi_1 \texttt{ else } \Pi_2 \texttt{ endif})^{tr} =_{def} \texttt{if } F \texttt{ then } \{\Pi_1^{tr}\} \texttt{ else } \{\Pi_2^{tr}\}$$
$$(\texttt{par } \Pi_1 \| \dots \| \Pi_n \texttt{ endpar})^{tr} =_{def} \Pi_1^{tr}; \dots; \Pi_n^{tr}$$

est insuffisante ! Par exemple le programme :

$$\texttt{par } x := y \| y := x \texttt{ endpar}$$

échange les valeurs de x et y, alors que sa traduction

$$x := y; y := x$$

écrase la valeur de x par la valeur de y.

Il faut des variables temporaires !

$$
\begin{array}{l}
\texttt{if } F_1 \texttt{ then par} \\
\quad f_1^1(\vec{t}_1^1) := t_1^1 \\
\quad f_2^1(\vec{t}_2^1) := t_2^1 \\
\quad \vdots \\
\quad f_{m_1}^1(\vec{t}_{m_1}^1) := t_{m_1}^1 \\
\texttt{endpar else} \\
\vdots \\
\texttt{if } F_c \texttt{ then par} \\
\quad f_1^c(\vec{t}_1^c) := t_1^c \\
\quad f_2^c(\vec{t}_2^c) := t_2^c \\
\quad \vdots \\
\quad f_{m_c}^c(\vec{t}_{m_c}^c) := t_{m_c}^c \\
\texttt{endpar ... endif}
\end{array}
$$

$$
\begin{array}{l}
\texttt{if } v_{F_1} \texttt{ then } \{ \\
\quad f_1^1(\vec{v}_{t_1^1}) := v_{t_1^1}; \\
\quad f_2^1(\vec{v}_{t_2^1}) := v_{t_2^1}; \\
\quad \vdots \\
\quad f_{m_1}^1(\vec{v}_{t_{m_1}^1}) := v_{t_{m_1}^1}; \\
\} \texttt{ else } \{ \\
\vdots \\
\texttt{if } v_{F_c} \texttt{ then } \{ \\
\quad f_1^c(\vec{v}_{t_1^c}) := v_{t_1^c}; \\
\quad f_2^c(\vec{v}_{t_2^c}) := v_{t_2^c}; \\
\quad \vdots \\
\quad f_{m_c}^c(\vec{v}_{t_{m_c}^c}) := v_{t_{m_c}^c}; \\
\}; \ldots \}; \texttt{end}
\end{array}
$$

Il reste à :

Il reste à :

- Initialiser les variables $\{v_t \mid t \in Read(\Pi)\}$ par $\vec{v}_t := \vec{t}$.

Il reste à :

- Initialiser les variables $\{v_t \mid t \in Read(\Pi)\}$ par $\vec{v}_t := \vec{t}$.
- Rendre la dilatation temporelle uniforme avec des `skip`.

Il reste à :

- Initialiser les variables $\{v_t \mid t \in Read(\Pi)\}$ par $\vec{v_t} := \vec{t}$.
- Rendre la dilatation temporelle uniforme avec des `skip`.

Nous obtenons donc le programme P_Π traduisant une étape de Π :

Proposition (Traduction d'une étape d'ASM)

$$\Delta(P_\Pi, X)|_{\mathcal{L}(\Pi)} = \Delta(\Pi, X|_{\mathcal{L}(\Pi)})$$

$$time(P_\Pi, X) = r + c + m = t_\Pi$$

Il reste à :

- Initialiser les variables $\{v_t \mid t \in Read(\Pi)\}$ par $\vec{v}_t := \vec{t}$.
- Rendre la dilatation temporelle uniforme avec des `skip`.

Nous obtenons donc le programme P_Π traduisant une étape de Π :

Proposition (Traduction d'une étape d'ASM)

$$\Delta(P_\Pi, X)|_{\mathcal{L}(\Pi)} = \Delta(\Pi, X|_{\mathcal{L}(\Pi)})$$

$$time(P_\Pi, X) = r + c + m = t_\Pi$$

- r est le nombre de termes lus par Π
- c est le nombre d'états reconnus par Π
- m est le degré de parallélisme de Π

Plan

Un programme P_{φ_Π} durant assez longtemps ?

Un programme P_{φ_Π} durant assez longtemps ?

- φ_Π quelconque :

$$\texttt{while } \textit{true } \{\}$$

Un programme P_{φ_Π} durant assez longtemps ?

- φ_Π quelconque :

$$\texttt{while } \mathit{true}\ \{\}$$

- $\varphi_\Pi \in \mathcal{PR}$, il existe $P \in \texttt{Loop}$ la calculant, donc :

$$r := P(|X|);$$
$$\texttt{loop } r\ \{\}$$

Un programme P_{φ_Π} durant assez longtemps ?

- φ_Π quelconque :

$$\texttt{while } \mathit{true}\ \{\}$$

- $\varphi_\Pi \in \mathcal{PR}$, il existe $P \in \texttt{Loop}$ la calculant, donc :

$$r := P(|X|);$$
$$\texttt{loop } r\ \{\}$$

- complexité $\mathcal{P}ol$: $\varphi_\Pi(|X|) = \sum_{0 \leq i \leq d} a_i |X|^i \leq c \times |X|^n$

Un programme P_{φ_Π} durant assez longtemps ?

- φ_Π quelconque :

$$\texttt{while } \mathit{true}\ \{\}$$

- $\varphi_\Pi \in \mathcal{PR}$, il existe $P \in \texttt{Loop}$ la calculant, donc :

$$r := P(|X|);$$
$$\texttt{loop } r\ \{\}$$

- complexité $\mathcal{P}ol$: $\varphi_\Pi(|X|) = \sum_{0 \leq i \leq d} a_i |X|^i \leq c \times |X|^n$

$$\texttt{loop } c$$
$$\quad \texttt{loop } |X|$$
$$\quad\quad \ddots \text{ } (n \text{ fois})$$
$$\quad\quad\quad \texttt{loop } |X|\ \{\}$$

Détecter la fin de l'exécution ?

Détecter la fin de l'exécution ?

$$F_{\Pi} =_{def} \bigwedge_{t \in Read(\Pi)} v_t = t$$

Détecter la fin de l'exécution ?

$$F_\Pi =_{def} \bigwedge_{t \in Read(\Pi)} v_t = t$$

F_Π ne devient vraie qu'au bout d'exactement $time(\Pi, X|_{\mathcal{L}(\Pi)}) + 1$ répétitions de P_Π.

Détecter la fin de l'exécution ?

$$F_\Pi =_{def} \bigwedge_{t \in Read(\Pi)} v_t = t$$

F_Π ne devient vraie qu'au bout d'exactement $time(\Pi, X|_{\mathcal{L}(\Pi)}) + 1$ répétitions de P_Π. Le programme-complexité est $P = P_{\varphi_\Pi}^{F_\Pi}$

Détecter la fin de l'exécution ?

$$F_\Pi =_{def} \bigwedge_{t \in Read(\Pi)} v_t = t$$

F_Π ne devient vraie qu'au bout d'exactement $time(\Pi, X|_{\mathcal{L}(\Pi)}) + 1$ répétitions de P_Π. Le programme-complexité est $P = P_{\varphi_\Pi}^{F_\Pi}$, où :

Définition (Le programme s'arrêtant quand F_Π est vraie)

$$\begin{aligned}
(\texttt{end})^{F_\Pi} &=_{def} \texttt{end} \\
(c; P)^{F_\Pi} &=_{def} (c)^{F_\Pi}; \\
&\qquad \texttt{if } \neg F_\Pi \texttt{ then } \{P^{F_\Pi}\} \\
(ft_1 \dots t_k := t_0)^{F_\Pi} &=_{def} ft_1 \dots t_k := t_0 \\
(\texttt{while } F \ \{P\})^{F_\Pi} &=_{def} \texttt{while } (F \wedge \neg F_\Pi) \\
&\qquad \texttt{if } \neg F_\Pi \texttt{ then } \{P^{F_\Pi}\} \\
(\texttt{loop } n \ \{P\})^{F_\Pi} &=_{def} \texttt{loop } n \texttt{ except } F_\Pi \\
&\qquad \texttt{if } \neg F_\Pi \texttt{ then } \{P^{F_\Pi}\}
\end{aligned}$$

Plan

Pour simuler Π nous avons :

Pour simuler Π nous avons :

- un programme $P = P^{F_\Pi}_{\varphi_\Pi}$ durant autant que l'exécution de Π

Pour simuler Π nous avons :

- un programme $P = P_{\varphi_\Pi}^{F_\Pi}$ durant autant que l'exécution de Π
- un programme $Q = P_\Pi$ simulant une étape de Π en t_Π étapes

Pour simuler Π nous avons :

- un programme $P = P_{\varphi_\Pi}^{F_\Pi}$ durant autant que l'exécution de Π
- un programme $Q = P_\Pi$ simulant une étape de Π en t_Π étapes

Définition (Insertion de Q après chaque commande de P)

$$\begin{aligned}
\texttt{end}[Q] &=_{def} \texttt{end} \\
(c; P)[Q] &=_{def} (c)[Q]; P[Q] \\
(ft_1 \dots t_k := t_0)[Q] &=_{def} ft_1 \dots t_k := t_0; Q \\
(\texttt{if } F \texttt{ then } \{P_1\} \texttt{ else } \{P_2\})[Q] &=_{def} \texttt{if } F \texttt{ then } \{Q; P_1[Q]\} \\
&\qquad\qquad \texttt{else } \{Q; P_2[Q]\} \\
(\texttt{while } F \ \{P\})[Q] &=_{def} \texttt{while } F \ \{Q; P[Q]\} \\
(\texttt{loop } n \texttt{ except } F \ \{P\})[Q] &=_{def} \texttt{loop } n \texttt{ except } F \ \{Q; P[Q]\}
\end{aligned}$$

Il nous reste à :

Il nous reste à :

- Initialiser les $\{v_t \mid t \in \mathit{Read}(\Pi)\}$ contenus dans F_Π, en ajoutant un P_Π au début du programme.

Il nous reste à :

- Initialiser les $\{v_t \mid t \in Read(\Pi)\}$ contenus dans F_Π, en ajoutant un P_Π au début du programme.
- Uniformiser le temps d'arrêt, en comptant dans i_{end} le nombre d'étapes faites après que F_Π soit vraie

Il nous reste à :

- Initialiser les $\{v_t \mid t \in Read(\Pi)\}$ contenus dans F_Π, en ajoutant un P_Π au début du programme.
- Uniformiser le temps d'arrêt, en comptant dans i_{end} le nombre d'étapes faites après que F_Π soit vraie

Le programme final :

$$P_\Pi; P^{F_\Pi}_{\varphi_\Pi}[\texttt{if } \neg F_\Pi \texttt{ then } \{P_\Pi\} \texttt{ else } \{i_{end} := i_{end} + 3\}]; \texttt{skip } i_{end} \to max_{end}$$

Il nous reste à :

- Initialiser les $\{v_t \mid t \in Read(\Pi)\}$ contenus dans F_Π, en ajoutant un P_Π au début du programme.
- Uniformiser le temps d'arrêt, en comptant dans i_{end} le nombre d'étapes faites après que F_Π soit vraie

Le programme final :

$$P_\Pi; P'^{F_\Pi}_{\varphi_\Pi}[\texttt{if } \neg F_\Pi \texttt{ then } \{P_\Pi\} \texttt{ else } \{i_{end} := i_{end} + 3\}]; \texttt{skip } i_{end} \to max_{end}$$

Théorème (Partie 2)

- `While` *simule* $\texttt{ASM}$
- `LoopC` *simule* $\texttt{ASM}_{\mathcal{PR}}$
- $\texttt{LoopC}_{\texttt{stat}}$ *simule* $\texttt{ASM}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi_\Pi)$

Théorème

- $\texttt{While} \simeq \texttt{Algo}$
- $\texttt{LoopC} \simeq \texttt{Algo}_{\mathcal{PR}}$ *si les opérations sont en espace* $\mathcal{PR}$
- $\texttt{LoopC}_{\texttt{stat}} \simeq \texttt{Algo}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi)$

Théorème

- $\texttt{While} \simeq \texttt{Algo}$
- $\texttt{LoopC} \simeq \texttt{Algo}_{\mathcal{PR}}$ *si les opérations sont en espace* $\mathcal{PR}$
- $\texttt{LoopC}_{\texttt{stat}} \simeq \texttt{Algo}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi)$

- Corollaire : forme normale des programmes impératifs

Théorème

- $\texttt{While} \simeq \texttt{Algo}$
- $\texttt{LoopC} \simeq \texttt{Algo}_{\mathcal{PR}}$ *si les opérations sont en espace* $\mathcal{PR}$
- $\texttt{LoopC}_{\texttt{stat}} \simeq \texttt{Algo}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi)$

- Corollaire : forme normale des programmes impératifs
- Perdre du temps ? !

Théorème

- $\texttt{While} \simeq \texttt{Algo}$
- $\texttt{LoopC} \simeq \texttt{Algo}_{\mathcal{PR}}$ *si les opérations sont en espace* $\mathcal{PR}$
- $\texttt{LoopC}_{\texttt{stat}} \simeq \texttt{Algo}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi)$

- Corollaire : forme normale des programmes impératifs
- Perdre du temps ? ! Relâcher la définition...

Théorème

- $\texttt{While} \simeq \texttt{Algo}$
- $\texttt{LoopC} \simeq \texttt{Algo}_{\mathcal{PR}}$ *si les opérations sont en espace* $\mathcal{PR}$
- $\texttt{LoopC}_{\texttt{stat}} \simeq \texttt{Algo}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi)$

- Corollaire : forme normale des programmes impératifs
- Perdre du temps ? ! Relâcher la définition...
- Les constantes devraient compter !

Théorème

- $\texttt{While} \simeq \texttt{Algo}$
- $\texttt{LoopC} \simeq \texttt{Algo}_{\mathcal{PR}}$ *si les opérations sont en espace* $\mathcal{PR}$
- $\texttt{LoopC}_{\texttt{stat}} \simeq \texttt{Algo}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi)$

- Corollaire : forme normale des programmes impératifs
- Perdre du temps ? ! Relâcher la définition...
- Les constantes devraient compter ! $\texttt{Imp}^{+}$

Théorème

- $\texttt{While} \simeq \texttt{Algo}$
- $\texttt{LoopC} \simeq \texttt{Algo}_{\mathcal{PR}}$ *si les opérations sont en espace* $\mathcal{PR}$
- $\texttt{LoopC}_{\texttt{stat}} \simeq \texttt{Algo}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi)$

- Corollaire : forme normale des programmes impératifs
- Perdre du temps ? ! Relâcher la définition...
- Les constantes devraient compter ! $\texttt{Imp}^{+}$, speed-up

Théorème

- $\texttt{While} \simeq \texttt{Algo}$
- $\texttt{LoopC} \simeq \texttt{Algo}_{\mathcal{PR}}$ *si les opérations sont en espace* $\mathcal{PR}$
- $\texttt{LoopC}_{\texttt{stat}} \simeq \texttt{Algo}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi)$

- Corollaire : forme normale des programmes impératifs
- Perdre du temps ? ! Relâcher la définition...
- Les constantes devraient compter ! $\texttt{Imp}^{+}$, speed-up
- Seul LoopC a des contraintes de structure de données

Théorème

- $\texttt{While} \simeq \texttt{Algo}$
- $\texttt{LoopC} \simeq \texttt{Algo}_{\mathcal{PR}}$ *si les opérations sont en espace* $\mathcal{PR}$
- $\texttt{LoopC}_{\texttt{stat}} \simeq \texttt{Algo}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi)$

- Corollaire : forme normale des programmes impératifs
- Perdre du temps ? ! Relâcher la définition...
- Les constantes devraient compter ! $\texttt{Imp}^{+}$, speed-up
- Seul $\texttt{LoopC}$ a des contraintes de structure de données
- $\texttt{LoopC}_{\texttt{stat}}$ a une caractérisation du degré !

Théorème

- $\texttt{While} \simeq \texttt{Algo}$
- $\texttt{LoopC} \simeq \texttt{Algo}_{\mathcal{PR}}$ *si les opérations sont en espace* $\mathcal{PR}$
- $\texttt{LoopC}_{\texttt{stat}} \simeq \texttt{Algo}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi)$

- Corollaire : forme normale des programmes impératifs
- Perdre du temps ? ! Relâcher la définition...
- Les constantes devraient compter ! $\texttt{Imp}^+$, speed-up
- Seul $\texttt{LoopC}$ a des contraintes de structure de données
- $\texttt{LoopC}_{\texttt{stat}}$ a une caractérisation du degré ! Mais peu naturel...

Théorème

- $\texttt{While} \simeq \texttt{Algo}$
- $\texttt{LoopC} \simeq \texttt{Algo}_{\mathcal{PR}}$ *si les opérations sont en espace* $\mathcal{PR}$
- $\texttt{LoopC}_{\texttt{stat}} \simeq \texttt{Algo}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi)$

- Corollaire : forme normale des programmes impératifs
- Perdre du temps ? ! Relâcher la définition...
- Les constantes devraient compter ! $\texttt{Imp}^+$, speed-up
- Seul $\texttt{LoopC}$ a des contraintes de structure de données
- $\texttt{LoopC}_{\texttt{stat}}$ a une caractérisation du degré ! Mais peu naturel...
- Autres modèles ?

Théorème

- $\texttt{While} \simeq \texttt{Algo}$
- $\texttt{LoopC} \simeq \texttt{Algo}_{\mathcal{PR}}$ *si les opérations sont en espace* $\mathcal{PR}$
- $\texttt{LoopC}_{\texttt{stat}} \simeq \texttt{Algo}_{\mathcal{P}ol}$ *avec* $depth(P) = deg(\varphi)$

- Corollaire : forme normale des programmes impératifs
- Perdre du temps ? ! Relâcher la définition...
- Les constantes devraient compter ! $\texttt{Imp}^+$, speed-up
- Seul $\texttt{LoopC}$ a des contraintes de structure de données
- $\texttt{LoopC}_{\texttt{stat}}$ a une caractérisation du degré ! Mais peu naturel...
- Autres modèles ? Moschovakis (spécification)...