

# Calculabilité

P. Courtieu

21 mars 2013





# Introduction

- ▶ Aujourd'hui :  $\lambda$ -calcul non typé
- ▶ Dû à Alonzo Church (années 30)
- ▶ Contemporain des machines de Turing
- ▶ Au moins aussi fondateur :
  - ▶ Modèle de calcul
  - ▶ Origine de la programmation fonctionnelle
  - ▶ Dédution formelle (Coq)

## Références

- ▶ H. Barendregt, The Lambda-Calculus, volume 103, Elsevier Science Publishing Company, Amsterdam, 1984.
- ▶ Jean-Louis Krivine, Lambda-Calcul, types et modèles, Masson 1991.
- ▶ Wikipedia (version française bien, version anglaise plus complète)

## Qu'est-ce qu'une fonction ?

- ▶ Purement syntaxique
- ▶ Expression « attendant » un argument
- ▶ Calcul = *substituer* l'argument appliqué
- ▶ C'est tout !
- ▶ (mauvais<sup>1</sup>) exemple :

Si  $\text{inc}(x) = x + 1$  alors  $\text{inc}(3) \equiv 3 + 1$ .

---

1. « + » et « 1 » n'existent pas, Nous les *coderons* plus loin (*Entiers de Church*).

## Comment écrire une fonction ?

▶  $x + 1$ ? **NON** : **Expression contenant  $x \neq$  fonction sur  $x$ .**

▶  $x \mapsto x + 1$ ? **OUI** :

$\lambda x.(x + 1)$

▶ Application :

$f e$

# Exemples

$$\begin{aligned}(\lambda x.(x + 1)) 3 &\rightarrow 3 + 1 \\(\lambda x.(\lambda y.(x + y))) 3 &\rightarrow \lambda y.(3 + y)\end{aligned}$$

$$\begin{aligned}(\text{fun } x \rightarrow x+1) 3 \\(\text{fun } x \rightarrow \text{fun } y \rightarrow x+y) 3\end{aligned}$$

# Syntaxe

Grammaire extrêmement minimale :

$e ::=$	$x$	Variable
	$\lambda x.e$	$\lambda$ – « Abstraction »
	$e e$	Application fonction/argument

Ex :

- ▶  $\lambda x.x$ ,  $\lambda x.y$ ,  $\lambda x.x x$ ,  $\lambda x.(x x)$ ,  $(\lambda x.x) x$ ,
- ▶  $\lambda x.(\lambda y.(x x))y$
- ▶ Terme = arbre
- ▶ Parenthèses pour « désambiguer » l'écriture linéaire.

# Parenthèses

- ▶ Application associée à gauche.
  - ▶  $x y z = (x y) z$  (et pas  $x (y z)$ )
- ▶ Application  $>$  abstraction
  - ▶  $\lambda x.x x = \lambda x.(x x)$  (et pas  $(\lambda x.x) x$ )
  - ▶  $\lambda x.x y x = \lambda x.((x y) x)$
- ▶  $\lambda x.\lambda y.\lambda z.e = \lambda x.(\lambda y.(\lambda z.e))$  (pas d'autre choix)

## Abus de notation (que j'éviterai autant que possible)

- ▶  $\lambda xyz.e = \lambda x.\lambda y.\lambda z.e$
- ▶  $xy = x y$  Si clair que  $xy$  pas un nom de variable

## Réduction / calcul

Une seule règle de calcul :

$$(\lambda x.t) u \rightarrow_{\beta} t[x \leftarrow u]$$

$t[x \leftarrow u] \equiv$   $t$  « dans lequel toutes les occurrences de  $x$  sont remplacées par  $u$  »

Ex :

## Réduction / calcul

Une seule règle de calcul :

$$(\lambda x.t) u \rightarrow_{\beta} t[x \leftarrow u]$$

$t[x \leftarrow u] \equiv$   $t$  « dans lequel toutes les occurrences de  $x$  sont remplacées par  $u$  »

- Ex :
- ▶  $(\lambda x.(x y)) z \rightarrow_{\beta} z y$
  - ▶  $(\lambda x.x y) z \rightarrow_{\beta} z y$

## Réduction / calcul

Une seule règle de calcul :

$$(\lambda x.t) u \rightarrow_{\beta} t[x \leftarrow u]$$

$t[x \leftarrow u] \equiv$   $t$  « dans lequel toutes les occurrences de  $x$  sont remplacées par  $u$  »

- Ex :
- ▶  $(\lambda x.(x y)) z \rightarrow_{\beta} z y$
  - ▶  $(\lambda x.x y) z \rightarrow_{\beta} z y$
  - ▶  $(\lambda x.x y x) z \rightarrow_{\beta} z y z$

## Réduction / calcul

Une seule règle de calcul :

$$(\lambda x.t) u \rightarrow_{\beta} t[x \leftarrow u]$$

$t[x \leftarrow u] \equiv$   $t$  « dans lequel toutes les occurrences de  $x$  sont remplacées par  $u$  »

Ex :  $\blacktriangleright (\lambda x.x y (\lambda x.x t)) z \rightarrow_{\beta} (x y (\lambda x.x t))[x \leftarrow z]$

## Réduction / calcul

Une seule règle de calcul :

$$(\lambda x.t) u \rightarrow_{\beta} t[x \leftarrow u]$$

$t[x \leftarrow u] \equiv$   $t$  « dans lequel toutes les occurrences **libres** de  $x$  sont remplacées par  $u$  »

Ex :  $\blacktriangleright$   $(\lambda x.x y (\lambda x.x t)) z \rightarrow_{\beta} (x y (\lambda x.x t))[x \leftarrow z]$   
 $\equiv z y (\lambda x.x t)$

## Réduction / collision de variable

$$(\lambda x. \lambda y. x y) y$$

## Réduction / collision de variable

$$(\lambda x. \lambda \underline{y}. x \underline{y}) y$$

## Réduction / collision de variable

$(\lambda x. \lambda y. x y) \underline{y}$

## Réduction / collision de variable

$$(\lambda x. \lambda y. x y) y \rightarrow_{\beta} (\lambda y. x y)[x \leftarrow y]$$

## Réduction / collision de variable

$(\lambda x. \lambda y. x y) y \rightarrow_{\beta} (\lambda y. x y)[x \leftarrow y] \equiv \lambda y. y y$  **NON**, confusion de deux  $y$  différents

## Réduction / collision de variable

$$\begin{aligned} & (\lambda x. \lambda y. x y) y \\ & \equiv (\lambda x. \lambda z. x z) y \end{aligned}$$

## Réduction / collision de variable

$$\begin{aligned} & (\lambda x. \lambda y. x y) y \\ & \equiv (\lambda x. \lambda z. x z) y \rightarrow_{\beta} (\lambda z. x z)[x \leftarrow y] \end{aligned}$$

## Réduction / collision de variable

$$\begin{aligned} & (\lambda x. \lambda y. x y) y \\ & \equiv (\lambda x. \lambda z. x z) y \rightarrow_{\beta} (\lambda z. x z)[x \leftarrow y] \equiv \lambda z. y z \quad \text{OUI} \end{aligned}$$

## Réduction / collision de variable

$$\begin{aligned} & (\lambda x. \lambda y. x y) y \\ & \equiv (\lambda x. \lambda z. x z) y \rightarrow_{\beta} (\lambda z. x z)[x \leftarrow y] \equiv \lambda z. y z \text{ OUI} \end{aligned}$$

- ▶ En revanche :  $(\lambda x. x y) y \rightarrow_{\beta} y y$  OK
- ▶  $\alpha$ -équivalence : renommage des variables *liées*

$$\lambda x. t \equiv_{\alpha} \lambda z. (t[x \leftarrow z])$$

## Variables libres

$FV(t)$  : Ensemble des variables ayant une occurrence libre dans  $t$  :

$$\begin{aligned}FV(x) &= \{x\} \\FV(t_1 t_2) &= FV(t_1) \cup FV(t_2) \\FV(\lambda x.t) &= FV(t) \setminus \{x\}\end{aligned}$$

(Définition par induction)

## Substitutions des variables libres (sans capture)

Par *induction* sur les termes :

$$x[x \leftarrow t] \equiv t$$

$$x[y \leftarrow t] \equiv x \quad (\text{si } x \neq y)$$

$$(t_1 t_2)[x \leftarrow t] \equiv (t_1[x \leftarrow t]) (t_2[x \leftarrow t])$$

$$(\lambda x.t)[x \leftarrow u] \equiv t$$

$$(\lambda x.t)[y \leftarrow u] \equiv \lambda x.(t[y \leftarrow u])$$

## Substitutions des variables libres (sans capture)

Par *induction* sur les termes :

$$x[x \leftarrow t] \equiv t$$

$$x[y \leftarrow t] \equiv x \quad (\text{si } x \neq y)$$

$$(t_1 \ t_2)[x \leftarrow t] \equiv (t_1[x \leftarrow t]) (t_2[x \leftarrow t])$$

$$(\lambda x.t)[x \leftarrow u] \equiv t$$

$$(\lambda x.t)[y \leftarrow u] \equiv \lambda x.(t[y \leftarrow u]) \quad \text{si } x \neq y \text{ et si } x \notin FV(u)$$

## $\beta$ -équivalence

$$t =_{\beta} u \text{ ssi } t \leftrightarrow_{\beta}^* u$$

- ▶  $(\lambda x.(\lambda y.x) z) t \rightarrow_{\beta} (\lambda y.t) z$   
 $(\lambda x.(\lambda y.x) z) t \rightarrow_{\beta} (\lambda x.x) t$
- ▶  $(\lambda x.(\lambda y.x) z) t =_{\beta} (\lambda y.t) z =_{\beta} (\lambda x.x) t$

## Propriétés du $\lambda$ -calcul

- ▶ Unicité de la forme normale
- ▶ Pas de terminaison systématique
- ▶ Stratégies de réduction
- ▶ Terminaison sur les termes « bien typés »
- ▶ Vous verrez ça plus tard dans le cours

## Codage des booléens

Idée : true = « THEN », false = « ELSE »

- ▶  $TRUE := \lambda x.\lambda y.x$                        $FALSE := \lambda x.\lambda y.y$   
 $TRUE\ t_1\ t_2 \rightarrow_{\beta} t_1$                        $FALSE\ t_1\ t_2 \rightarrow_{\beta} t_2$
- ▶  $IFTHENELSE := \lambda p.\lambda a.\lambda b.p\ a\ b$   
 $IFTHENELSE\ TRUE\ t_1\ t_2 \rightarrow_{\beta} TRUE\ t_1\ t_2 \rightarrow_{\beta} t_1$   
 $IFTHENELSE\ FALSE\ t_1\ t_2 \rightarrow_{\beta} FALSE\ t_1\ t_2 \rightarrow_{\beta} t_2$

Exercice : AND, OR et NOT.

## Exercice : Codage de AND et OR

▶  $AND := ?$

▶  $OR := ?$

## Exercice : Codage de AND et OR

▶  $AND := ?$

« p et q » = « si p alors q sinon false »

▶  $OR := ?$

## Exercice : Codage de AND et OR

►  $AND := ?$

« p et q » = « si p alors q sinon false »

« p et q » = « si p alors q sinon p »

►  $OR := ?$

## Exercice : Codage de AND et OR

►  $AND := ?$

« p et q » = « si p alors q sinon false »

« p et q » = « si p alors q sinon p »

$\lambda p.\lambda q. IFTHENELSE\ p\ q\ p$

►  $OR := ?$

## Exercice : Codage de AND et OR

- ▶  $AND :=$ 
  - « p et q » = « si p alors q sinon false »
  - « p et q » = « si p alors q sinon p »
  - $\lambda p.\lambda q. IFTHENELSE\ p\ q\ p$
  - $\lambda p.\lambda q. p\ q\ p$
  
- ▶  $OR := ?$

## Exercice : Codage de AND et OR

▶  $AND := \lambda p.\lambda q.p\ q\ p$

▶  $OR := ?$

## Exercice : Codage de AND et OR

▶  $AND := \lambda p.\lambda q.p\ q\ p$

▶  $OR := ?$

« p ou q » = « si p alors true sinon q »

## Exercice : Codage de AND et OR

►  $AND := \lambda p.\lambda q.p\ q\ p$

►  $OR := ?$

« p ou q » = « si p alors true sinon q »

« p ou q » = « si p alors p sinon q »

## Exercice : Codage de AND et OR

►  $AND := \lambda p.\lambda q.p\ q\ p$

►  $OR := ?$

« p ou q » = « si p alors true sinon q »

« p ou q » = « si p alors p sinon q »

$\lambda p.\lambda q.IFTHENELSE\ p\ p\ q$

## Exercice : Codage de AND et OR

►  $AND := \lambda p.\lambda q.p\ q\ p$

►  $OR := ?$

« p ou q » = « si p alors true sinon q »

« p ou q » = « si p alors p sinon q »

$\lambda p.\lambda q.IFTHENELSE\ p\ p\ q$

$\lambda p.\lambda q.p\ p\ q$

## Exercice : Codage de AND et OR

▶  $AND := \lambda p.\lambda q.p\ q\ p$

▶  $OR := \lambda p.\lambda q.p\ p\ q$

# Exercice : Codage de NOT

►  $NOT := ?$

## Exercice : Codage de NOT

- ▶  $NOT := ?$  Doit transformer  $\lambda x.\lambda y.x$  en  $\lambda x.\lambda y.y$  et vice-versa.

## Exercice : Codage de NOT

- ▶  $NOT := ?$  Doit transformer  $\lambda x.\lambda y.x$  en  $\lambda x.\lambda y.y$  et vice-versa.  
(?)  $(\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$

## Exercice : Codage de NOT

- ▶  $NOT := ?$  Doit transformer  $\lambda x.\lambda y.x$  en  $\lambda x.\lambda y.y$  et vice-versa.  
 $(?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$   
 $(\lambda\rho.?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$

## Exercice : Codage de NOT

- ▶  $NOT := ?$  Doit transformer  $\lambda x.\lambda y.x$  en  $\lambda x.\lambda y.y$  et vice-versa.

$$(?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$$

$$(\lambda\rho.?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$$

$$(\lambda\rho.\lambda x.\lambda y.?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$$

## Exercice : Codage de NOT

- ▶  $NOT := ?$  Doit transformer  $\lambda x.\lambda y.x$  en  $\lambda x.\lambda y.y$  et vice-versa.

$$(?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$$

$$(\lambda p.?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$$

$$(\lambda p.\lambda x.\lambda y.?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$$

$$(\lambda p.\lambda x.\lambda y.p \ y \ ?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y \text{ OK!}$$

## Exercice : Codage de NOT

- ▶  $NOT := ?$  Doit transformer  $\lambda x.\lambda y.x$  en  $\lambda x.\lambda y.y$  et vice-versa.

$$(?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$$

$$(\lambda p.?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$$

$$(\lambda p.\lambda x.\lambda y.?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$$

$$(\lambda p.\lambda x.\lambda y.p \ y \ ?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y \text{ OK!}$$

Vice versa :

$$(\lambda p.\lambda x.\lambda y.p \ y \ ?) (\lambda x.\lambda y.y) \rightarrow_{\beta} \lambda x.\lambda y.x$$

## Exercice : Codage de NOT

- ▶  $NOT := ?$  Doit transformer  $\lambda x.\lambda y.x$  en  $\lambda x.\lambda y.y$  et vice-versa.

$$(?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$$

$$(\lambda p.?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$$

$$(\lambda p.\lambda x.\lambda y.?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y$$

$$(\lambda p.\lambda x.\lambda y.p \ y \ ?) (\lambda x.\lambda y.x) \rightarrow_{\beta} \lambda x.\lambda y.y \text{ OK!}$$

Vice versa :

$$(\lambda p.\lambda x.\lambda y.p \ y \ ?) (\lambda x.\lambda y.y) \rightarrow_{\beta} \lambda x.\lambda y.x$$

$$(\lambda p.\lambda x.\lambda y.p \ y \ x) (\lambda x.\lambda y.y) \rightarrow_{\beta} \lambda x.\lambda y.x \text{ OK!}$$

## Exercice : Codage de NOT

►  $NOT := \lambda p.\lambda x.\lambda y.p\ y\ x$

## Codage des entiers (Church)

$n =$  fonction  $f \mapsto f^n$ , où  $f$  est une fonction

- ▶  $0 := \lambda f. \lambda x. x$
- ▶  $1 := \lambda f. \lambda x. f\ x$
- ▶  $2 := \lambda f. \lambda x. f\ (f\ x)$
- ▶  $3 := \lambda f. \lambda x. f\ (f\ (f\ x))$
- ▶ ...

## Ex : Codage du successeur

- ▶ Successeur ?

$$SUCC\ N \rightarrow_{\beta}^* N + 1$$

$$(\lambda f.\lambda x.f^n\ x)$$

## Ex : Codage du successeur

- Successeur ?

$$SUCC\ N \rightarrow_{\beta}^* N + 1$$

$$\begin{array}{ccc} (\lambda f. \lambda x. f^n x) & & \\ ((\lambda f. \lambda x. f^n x)\ f\ x) & \rightarrow_{\beta}^* & (f^n\ x) \end{array}$$

## Ex : Codage du successeur

- Successeur ?

$$SUCC\ N \rightarrow_{\beta}^* N + 1$$

$$\begin{array}{l}
 (\lambda f. \lambda x. f^n\ x) \\
 ((\lambda f. \lambda x. f^n\ x)\ f\ x) \rightarrow_{\beta}^* (f^n\ x) \\
 f\ ((\lambda f. \lambda x. f^n\ x)\ f\ x) \rightarrow_{\beta}^* f\ (f^n\ x)
 \end{array}$$

## Ex : Codage du successeur

- Successeur ?

$$SUCC\ N \rightarrow_{\beta}^* N + 1$$

$$\begin{array}{l}
 (\lambda f. \lambda x. f^n\ x) \\
 ((\lambda f. \lambda x. f^n\ x)\ f\ x) \rightarrow_{\beta}^* (f^n\ x) \\
 f\ ((\lambda f. \lambda x. f^n\ x)\ f\ x) \rightarrow_{\beta}^* f\ (f^n\ x) \\
 \lambda f. \lambda x. f\ ((\lambda f. \lambda x. f^n\ x)\ f\ x) \rightarrow_{\beta}^* \lambda f. \lambda x. f\ (f^n\ x)
 \end{array}$$

# Ex : Codage du successeur

- Successeur ?

$$SUCC\ N \rightarrow_{\beta}^* N + 1$$

$$\begin{array}{l}
 (\lambda f. \lambda x. f^n\ x) \\
 ((\lambda f. \lambda x. f^n\ x)\ f\ x) \rightarrow_{\beta}^* (f^n\ x) \\
 f\ ((\lambda f. \lambda x. f^n\ x)\ f\ x) \rightarrow_{\beta}^* f\ (f^n\ x) \\
 \lambda f. \lambda x. f\ ((\lambda f. \lambda x. f^n\ x)\ f\ x) \rightarrow_{\beta}^* \lambda f. \lambda x. f\ (f^n\ x) \\
 (\lambda n. \lambda f. \lambda x. f\ (n\ f\ x))\ N \rightarrow_{\beta}^* N + 1
 \end{array}$$

# Ex : Codage du successeur

► Successeur ?

$$SUCC := \lambda n. \lambda f. \lambda x. f (n f x)$$

$$\begin{array}{l}
 (\lambda f. \lambda x. f^n x) \\
 ((\lambda f. \lambda x. f^n x) f x) \rightarrow_{\beta}^* (f^n x) \\
 f ((\lambda f. \lambda x. f^n x) f x) \rightarrow_{\beta}^* f (f^n x) \\
 \lambda f. \lambda x. f ((\lambda f. \lambda x. f^n x) f x) \rightarrow_{\beta}^* \lambda f. \lambda x. f (f^n x) \\
 (\lambda n. \lambda f. \lambda x. f (n f x)) N \rightarrow_{\beta}^* N + 1
 \end{array}$$

## Ex : Codage du successeur

- ▶ Successeur ?

$$SUCC := \lambda n. \lambda f. \lambda x. f (n f x)$$

- ▶ Addition ?

## Ex : Codage de l'addition

?

## Ex : Codage de l'addition

$$\lambda n.\lambda p.\lambda f.\lambda x.n f (p f x).$$

# Codage du test à zéro

?

## Codage du test à zéro

$$\text{ISZERO} := \lambda n.n (\lambda x.FALSE) TRUE$$

## Ex : Codage de la multiplication

?

## Ex : Codage de la multiplication

$$\lambda n.\lambda p.\lambda f.n (p f)$$

## aires et listes

$PAIR := \lambda x. \lambda y. \lambda f. f \ x \ y$

$FIRST := \lambda p. p \ TRUE$

$SECOND := \lambda p. p \ FALSE$

$NIL := ?$

$NULL := ?$

## aires et listes

$PAIR := \lambda x. \lambda y. \lambda f. f \ x \ y$

$FIRST := \lambda p. p \ TRUE$

$SECOND := \lambda p. p \ FALSE$

$NIL := \lambda x. TRUE$

$NULL := \lambda p. p \ (\lambda x. \lambda y. FALSE)$

# Récursion

- ▶ Appliquer une fonction à elle-même
- ▶ Semble (à tort) impossible

# Récursion

- ▶ Appliquer une fonction à elle-même
- ▶ Semble (à tord) impossible
- ▶ Impossible sauf si argument d'une fonction

$$(\lambda f.f f)$$

# Récursion

- ▶ Appliquer une fonction à elle-même
- ▶ Semble (à tort) impossible
- ▶ Impossible sauf si argument d'une fonction
- ▶ Et qu'on l'applique elle-même

$$(\lambda f.f f)(\lambda f.f f)$$

# Récursion

- ▶ Appliquer une fonction à elle-même
- ▶ Semble (à tord) impossible
- ▶ Impossible sauf si argument d'une fonction
- ▶ Et qu'on l'applique elle-même

$$\begin{array}{l} (\lambda f.f f)(\lambda f.f f) \\ \rightarrow_{\beta} (\lambda f.f f)(\lambda f.f f) \end{array}$$

# Récursion

- ▶ Appliquer une fonction à elle-même
- ▶ Semble (à tord) impossible
- ▶ Impossible sauf si argument d'une fonction
- ▶ Et qu'on l'applique elle-même

$$\begin{aligned} & (\lambda f.f f)(\lambda f.f f) \\ \rightarrow_{\beta} & (\lambda f.f f)(\lambda f.f f) \\ \rightarrow_{\beta} & (\lambda f.f f)(\lambda f.f f) \\ \rightarrow_{\beta} & \dots \end{aligned}$$

# Récursion

- ▶ Appliquer une fonction à elle-même
- ▶ Semble (à tort) impossible
- ▶ Impossible sauf si argument d'une fonction
- ▶ Et qu'on l'applique elle-même

$$(\lambda f.1 + f f)$$

# Récursion

- ▶ Appliquer une fonction à elle-même
- ▶ Semble (à tord) impossible
- ▶ Impossible sauf si argument d'une fonction
- ▶ Et qu'on l'applique elle-même

$$(\lambda f.1 + f f)(\lambda f.1 + f f)$$

# Récursion

- ▶ Appliquer une fonction à elle-même
- ▶ Semble (à tord) impossible
- ▶ Impossible sauf si argument d'une fonction
- ▶ Et qu'on l'applique elle-même

$$\begin{array}{l} (\lambda f.1 + f f)(\lambda f.1 + f f) \\ \rightarrow_{\beta} 1 + (\lambda f.1 + f f)(\lambda f.1 + f f) \end{array}$$

# Récursion

- ▶ Appliquer une fonction à elle-même
- ▶ Semble (à tort) impossible
- ▶ Impossible sauf si argument d'une fonction
- ▶ Et qu'on l'applique elle-même

$$\begin{array}{l}
 (\lambda f.1 + f f)(\lambda f.1 + f f) \\
 \rightarrow_{\beta} \quad 1 + (\lambda f.1 + f f)(\lambda f.1 + f f) \\
 \rightarrow_{\beta} \quad 1 + 1 + (\lambda f.1 + f f)(\lambda f.1 + f f) \\
 \rightarrow_{\beta} \quad \dots
 \end{array}$$

# Factorielle

- ▶ Équation récursive :

$$F(n) = \text{if } n=0 \text{ then } 1 \text{ else } n \times F(n-1)$$

$$F = \lambda n. \text{ if } n=0 \text{ then } 1 \text{ else } n \times F (n-1)$$

- ▶ F auto-argument

$$\lambda F. \lambda n. \text{ if } n=0 \text{ then } 1 \text{ else } n \times F \text{ F } (n-1)$$

- ▶ Appliquée à elle-même

$$F :=$$

$$(\lambda F. \lambda n. \text{ if } n=0 \text{ then } 1 \text{ else } n \times F \text{ F } (n-1)) \quad \lambda F. \lambda n. \text{ if } n=0 \text{ then } 1 \text{ else } n \times F \text{ F } (n-1)$$

ou bien :

$$F := (\lambda x. x x) (\lambda F. \lambda n. \text{ if } n=0 \text{ then } 1 \text{ else } n \times F \text{ F } (n-1))$$

## Ex : Factorielle de 3

- ▶ On note :

$$G := \text{if } n=0 \text{ then } 1 \text{ else } n \times F F (n-1)$$

$$F := (\lambda F. \lambda n. G) (\lambda F. \lambda n. G)$$

- ▶ Calcul de  $F 3$

$$F 3 = (\lambda F. \lambda n. G) (\lambda F. \lambda n. G) 3$$

$$\rightarrow_{\beta} (\lambda n. G [F \leftarrow \lambda F. \lambda n. G]) 3$$

$$\rightarrow_{\beta} (\lambda n. \text{if } n=0 \text{ then } 1 \text{ else } n \times (\lambda F. \lambda n. G) (\lambda F. \lambda n. G) (n-1)) 3$$

$$\rightarrow_{\beta} (\text{if } 3=0 \text{ then } 1 \text{ else } 3 \times (\lambda F. \lambda n. G) (\lambda F. \lambda n. G) 2)$$

$$\rightarrow_{\beta}^* 3 \times (\lambda F. \lambda n. G) (\lambda F. \lambda n. G) 2$$

$$\equiv 3 \times F 2$$

## Ex : Factorielle de 2

► Calcul de  $F\ 2$

$$F\ 2 = (\lambda F.\lambda n.G) (\lambda F.\lambda n.G)\ 2$$

$$\rightarrow_{\beta} (\lambda n.G[F \leftarrow \lambda F.\lambda n.G])\ 2$$

$$\rightarrow_{\beta} (\lambda n.\text{if } n=0 \text{ then } 1 \text{ else } n \times (\lambda F.\lambda n.G) (\lambda F.\lambda n.G) (n-1))\ 2$$

$$\rightarrow_{\beta} (\text{if } 2=0 \text{ then } 1 \text{ else } 2 \times (\lambda F.\lambda n.G) (\lambda F.\lambda n.G) 1)$$

$$\rightarrow_{\beta}^* 2 \times (\lambda F.\lambda n.G) (\lambda F.\lambda n.G) 1$$

$$\equiv 2 \times F\ 1$$

## Ex : Factorielle de 1

► Calcul de  $F\ 1$

$$F\ 1 = (\lambda F.\lambda n.G) (\lambda F.\lambda n.G)\ 1$$

$$\rightarrow_{\beta} (\lambda n.G[F \leftarrow \lambda F.\lambda n.G])\ 1$$

$$\rightarrow_{\beta} (\lambda n.\text{if } n=0 \text{ then } 1 \text{ else } n \times (\lambda F.\lambda n.G) (\lambda F.\lambda n.G) (n-0))\ 1$$

$$\rightarrow_{\beta} (\text{if } 1=0 \text{ then } 1 \text{ else } 1 \times (\lambda F.\lambda n.G) (\lambda F.\lambda n.G) 0)$$

$$\rightarrow_{\beta}^* 1 \times (\lambda F.\lambda n.G) (\lambda F.\lambda n.G) 0$$

$$\equiv 1 \times F\ 0$$

## Ex : Factorielle de 0

► Calcul de  $F\ 0$

$$F\ 0 = (\lambda F.\lambda n.G) (\lambda F.\lambda n.G)\ 0$$

$$\rightarrow_{\beta} (\lambda n.G[F \leftarrow \lambda F.\lambda n.G])\ 0$$

$$\rightarrow_{\beta} (\lambda n.\text{if } n=0 \text{ then } 1 \text{ else } n \times (\lambda F.\lambda n.G) (\lambda F.\lambda n.G) (n-0))\ 0$$

$$\rightarrow_{\beta} (\text{if } 0=0 \text{ then } 1 \text{ else } 0 \times (\lambda F.\lambda n.G) (\lambda F.\lambda n.G) 0)$$

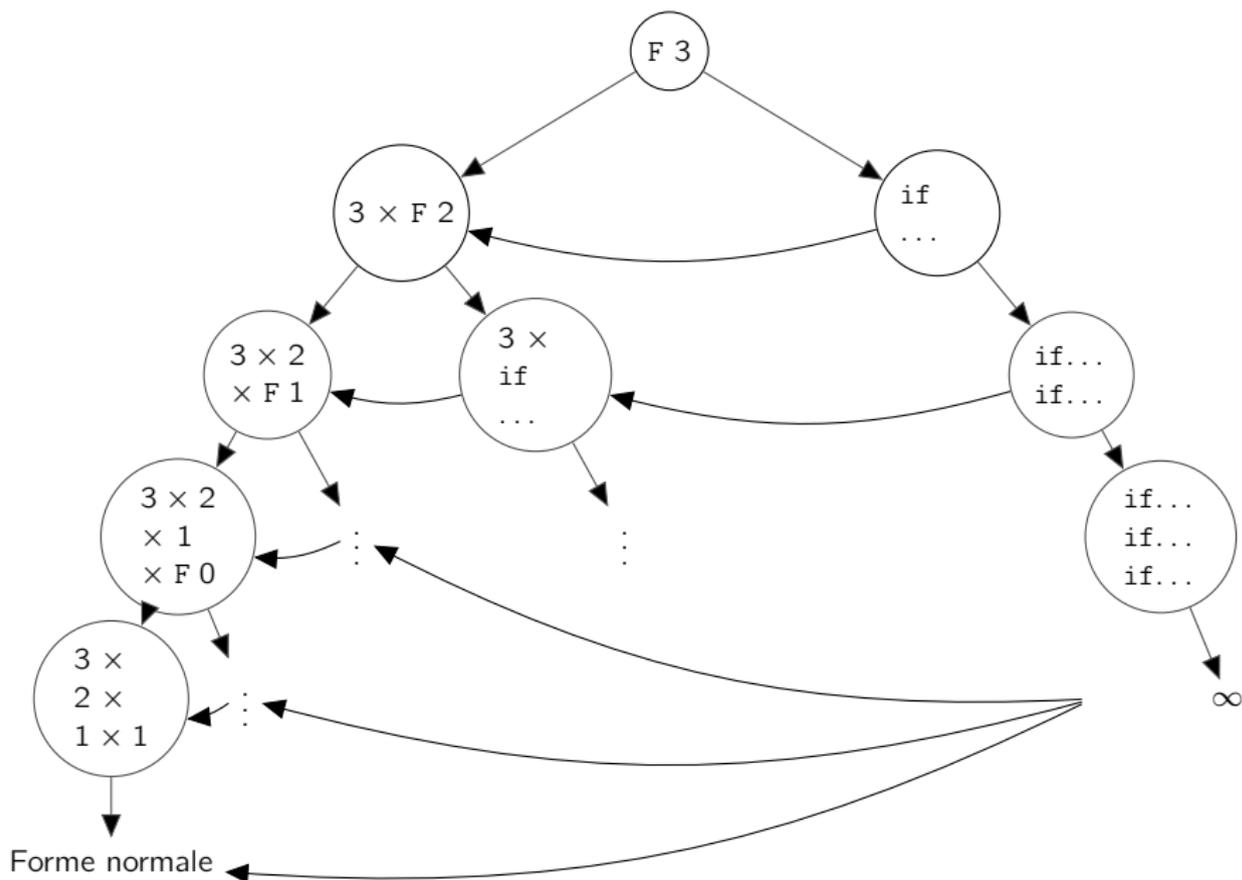
$$\rightarrow_{\beta}^* 1$$

## Quid si autre stratégie de réduction ?

$$\begin{aligned}
 F\ 3 &= (\lambda F.\lambda n.G) (\lambda F.\lambda n.G) 3 \\
 &\rightarrow_{\beta} (\lambda n.G[F \leftarrow \lambda F.\lambda n.G]) 3 \\
 &\equiv (\lambda n.\text{if } n=0 \text{ then } 1 \text{ else } n \times (\lambda F.\lambda n.G) (\lambda F.\lambda n.G) (n-1)) 3 \\
 &\rightarrow_{\beta} (\lambda n.\text{if } n=0 \text{ then } 1 \text{ else } n \times (\lambda n.G[F \leftarrow \lambda F.\lambda n.G]) (n-1)) 3 \\
 &\equiv (\lambda n.\text{if } n=0 \text{ then } 1 \text{ else } n \times \\
 &\quad (\lambda n.\text{if } n=0 \text{ then } 1 \text{ else } n \times (\lambda F.\lambda n.G) (\lambda F.\lambda n.G) (n-1)) \\
 &\quad (n-1)) 3
 \end{aligned}$$

... Calcul infini

# Confluence faible



## Récursion moins ad hoc ?

- ▶ Définition de la factorielle ad hoc
- ▶ Existe-t-il un mécanisme plus systématique ? Oui
- ▶ *Opérateurs de point fixe*
- ▶  $P$  t.q.  $P f =_{\beta} f (P f)$
- ▶  $P f x =_{\beta} (f (P f))x =_{\beta} (f (f (P f)))x \dots$
- ▶ Boucle infinie potentielle
- ▶ Stratégie : dépliage « suffisant » de la récursion

## Exemples d'opérateur de point fixe

$$\mathbf{Y} = \lambda f.((\lambda x.f (x x)) (\lambda x.f (x x)))$$

$$\begin{aligned}
 & \mathbf{Y} g \\
 = & \lambda f.((\lambda x.f (x x)) (\lambda x.f (x x))) g \\
 \rightarrow_{\beta} & (\lambda x.g (x x)) (\lambda x.g (x x)) \\
 \rightarrow_{\beta} & g ((\lambda x.g (x x)) (\lambda x.g (x x))) \\
 \leftarrow_{\beta} & g (\lambda f.((\lambda x.f (x x)) (\lambda x.f (x x))))g \\
 = & g (\mathbf{Y} g)
 \end{aligned}$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) 1 (n * (r (n - 1))))$$

$$(Y G) 3 =_{\beta} (G (G (G (Y G)))) 3$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) \ 1 \ (n * (r \ (n - 1))))$$

$$(Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$\left( G \left( G \left( \left( \lambda r. \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * r \ (n - 1)) \end{array} \right) \right) (Y \ G) \right) \right) \right) \right) 3$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) \ 1 \ (n * (r \ (n - 1)))) \\ (Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$\left( G \left( G \left( \left( \lambda r. \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * r \ (n - 1)) \end{array} \right) \right) (Y \ G) \right) \right) \right) \right) 3$$

$$\left( G \left( G \left( \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) \right) \right) \right) 3$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } ( \text{ISZERO } n) \ 1 \ (n * (r \ (n - 1)))) \\ (Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$\left( G \left( G \left( \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } ( \text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) \right) \right) \right) \ 3$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) \ 1 \ (n * (r \ (n - 1)))) \\ (Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$\left( G \left( G \left( \lambda n. \left( \text{IFTHENELSE } (\text{ISZERO } n) \right. \right. \right. \right. \\ \left. \left. \left. \left. \begin{array}{l} 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) \right) \right) \right) \right) \ 3$$

$$\left( G \left( \left( \left( \lambda r. \lambda n. \left( \text{IFTHENELSE } (\text{ISZERO } n) \right. \right. \right. \right. \right. \\ \left. \left. \left. \left. \begin{array}{l} 1 \\ (n * r \ (n - 1)) \end{array} \right) \right) \right) \right) \left( \lambda n. \left( \text{IFTHENELSE } (\text{ISZERO } n) \right. \right. \right. \\ \left. \left. \left. \left. \begin{array}{l} 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) \right) \right) \right) \right) \ 3$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) \ 1 \ (n * (r \ (n - 1))))$$

$$(Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$\left( G \left( \left( \left( \lambda r. \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * r \ (n - 1)) \end{array} \right) \right) \left( \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) \right) \right) \right) \right) 3$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) \ 1 \ (n * (r \ (n - 1)))) \\ (Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$\left( G \left( \left( \left( \lambda r. \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * r \ (n - 1)) \end{array} \right) \right) \left( \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) \right) \right) \right) \right) \ 3$$

$$\left( G \left( \left( \left( \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ \left( n * \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) \end{array} \right) \right) \right) \right) \right) \right) \ 3$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } ( \text{ISZERO } n) \ 1 \ (n * (r \ (n - 1))))$$

$$(Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$\left( G \left( \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } ( \text{ISZERO } n) \\ 1 \\ \left( n * \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } ( \text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) \end{array} \right) \right) \right) \right) \right) \ 3$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) \ 1 \ (n * (r \ (n - 1))))$$

$$(Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$\left( G \left( \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ n * \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) \end{array} \right) (n - 1) \right) \right) \right) \ 3$$

$$\left( \left( \lambda r. \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * r \ (n - 1)) \end{array} \right) \right) \left( \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ n * (\lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) ) \ (n - 1) \end{array} \right) \right) \right) \right) \ 3$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) \ 1 \ (n * (r \ (n - 1)))) \\ (Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$\left( \left( \lambda r. \lambda n. \left( \text{IFTHENELSE } (\text{ISZERO } n) \right. \right. \right. \\ \left. \left. \left. \begin{array}{l} 1 \\ (n * r \ (n - 1)) \end{array} \right) \right) \right) \left( \lambda n. \left( \text{IFTHENELSE } (\text{ISZERO } n) \right. \right. \\ \left. \left. \left. \begin{array}{l} 1 \\ \left( n * \left( \lambda n. \left( \text{IFTHENELSE } (\text{ISZERO } n) \right. \right. \right. \right. \\ \left. \left. \left. \begin{array}{l} 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) \right) \right) \right) \right) \right) \right) \right) \ 3$$











# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } ( \text{ISZERO } n ) \ 1 \ ( n * ( r \ ( n - 1 ) ) ) )$$

$$( Y \ G ) \ 3 =_{\beta} ( G \ ( G \ ( G \ ( Y \ G ) ) ) ) \ 3$$

$$3 * \left( \lambda n. \left( \begin{array}{l} \text{IFTHENELSE } ( \text{ISZERO } n ) \\ 1 \\ \left( n * ( \lambda n. \left( \begin{array}{l} \text{IFTHENELSE } ( \text{ISZERO } n ) \\ 1 \\ ( n * ( Y \ G ) \ ( n - 1 ) ) \end{array} \right) ) \ ( n - 1 ) \end{array} \right) \right) \right) \right) 2$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) \ 1 \ (n * (r \ (n - 1))))$$

$$(Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$3 * \left( \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ \left( n * \left( \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) \right) (n - 1) \end{array} \right) \right) \right) \right) 2$$

$$3 * \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } 2) \\ 1 \\ \left( 2 * \left( \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) \right) (2 - 1) \end{array} \right) \right)$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) \ 1 \ (n * (r \ (n - 1))))$$

$$(Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$3 * \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } 2) \\ 1 \\ \left( 2 * (\lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right)) \ (2 - 1) \right) \end{array} \right)$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) \ 1 \ (n * (r \ (n - 1))))$$

$$(Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$3 * \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } 2) \\ 1 \\ \left( 2 * (\lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right)) \right) (2 - 1) \end{array} \right)$$

$$3 * 2 * \lambda n. \left( \begin{array}{c} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) 1$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) \ 1 \ (n * (r \ (n - 1))))$$

$$(Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$3 * 2 * \lambda n. \left( \begin{array}{l} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) 1$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) \ 1 \ (n * (r \ (n - 1))))$$

$$(Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$3 * 2 * \lambda n. \left( \begin{array}{l} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) 1$$

$$3 * 2 * \left( \begin{array}{l} \text{IFTHENELSE } (\text{ISZERO } 1) \\ 1 \\ (1 * (Y \ G) \ (1 - 1)) \end{array} \right)$$

# Factorielle

$$G = \lambda r. \lambda n. (\text{IFTHENELSE } (\text{ISZERO } n) \ 1 \ (n * (r \ (n - 1)))) \\ (Y \ G) \ 3 =_{\beta} (G \ (G \ (G \ (Y \ G)))) \ 3$$

$$3 * 2 * \lambda n. \left( \begin{array}{l} \text{IFTHENELSE } (\text{ISZERO } n) \\ 1 \\ (n * (Y \ G) \ (n - 1)) \end{array} \right) 1$$

$$3 * 2 * \left( \begin{array}{l} \text{IFTHENELSE } (\text{ISZERO } 1) \\ 1 \\ (1 * (Y \ G) \ (1 - 1)) \end{array} \right)$$

$$3 * 2 * 1$$

## Exercice : réduction infinie

- ▶ Montrez que le calcul de factorielle de 3 peut ne pas terminer

## stratégies de réduction

- ▶ *Ordre applicatif* : à droite d'abord, en profondeur d'abord.
- ▶ *Appel par valeur* = à droite d'abord, en tête d'abord (Ocaml)
- ▶ *Normale* : à gauche d'abord, en tête d'abord (termine toujours, pas efficace)
- ▶ *Appel par nom* = Normale mais pas de réduction sous les  $\lambda$
- ▶ *Paresseux* = Normale mais partage des redex dupliqué (Haskell)

Normale : Termine toujours si il existe une forme normale

## Autres opérateurs de point fixe

- ▶ **Y** Boucle en stratégie appel par valeur
- ▶ **Y** Ok en paresseux
- ▶ **Z** =  $\lambda f.(\lambda x.f (\lambda y.x x y))(\lambda x.f (\lambda y.x x y))$
- ▶ **Z** OK en appel par valeur

## Op. de point fixe : dans quels calculs ?

L'ensemble des OPF est

- ▶ Infini (récursivement énumérablement) dans le  $\lambda$ -calcul non typé
- ▶ Vide dans le  $\lambda$ -calcul simplement typé (que vous verrez plus tard)
- ▶ Non vide dans certains  $\lambda$ -calculs à typage plus puissant (ex :  $F_\omega$ )



## Références

- ▶ Cours D. Pichardie (très bien)  
<http://www.irisa.fr/celtique/pichardie/teaching/L3/LOG/>
- ▶ Wikipedia (pas mal)

# Fonctions récursives primitives

- ▶ Décrire les fonctions sur  $\mathbb{N}$ 
  - ▶ Autres ensembles récursivement énumérables : codage
  - ▶ Définir la récurrence/récursion facilement
- ▶ Fonctions d'arité quelconque

$$\bigcup_{k \in \mathbb{N}} \mathcal{F}_k, \text{ où : } \mathcal{F}_k = \mathbb{N}^k \rightarrow \mathbb{N}$$

- ▶ Fonction de base + *Opérateurs* (composition, récursion, *minimisation*)

## Fonction récursive de base

▶ **Fonction zéro**  $0$  (d'arité 0) :

$$0() = 0$$

▶ **Fonction successeur**  $\sigma$  :

$$\sigma(n) = n + 1$$

▶ **Projections**  $\pi_i^k$  :

$$\pi_i^k(n_1, \dots, n_k) = n_i$$

# Exemples

▶ Fonction identité :  $Id = \pi_1^1$

$$Id(n) = \pi_1^1(n) = n$$

▶ Fonction  $fst = \pi_1^2$

$$fst(n, m) = \pi_1^2(n, m) = n$$

▶ Fonction  $snd = \pi_2^2$

$$snd(n, m) = \pi_2^2(n, m) = m$$

# Opérateurs de composition

- ▶  $f$  d'arité  $p$ ,
- ▶  $g_1, \dots, g_p$  d'arité  $q$ ,
- ▶ **Opérateur de composition**  $\mathcal{C}(f, (g_1, \dots, g_p))$  :

$$\mathcal{C}(f, (g_1, \dots, g_p))(n_1, \dots, n_q) \equiv f(g_1(n_1, \dots, n_q), \dots, g_p(n_1, \dots, n_q))$$

# Exemples

$$\begin{aligned} \mathcal{C}(f, (g_1, \dots, g_p)) (n_1, \dots, n_q) &\equiv f(g_1(n_1, \dots, n_q), \dots, g_p(n_1, \dots, n_q)) \\ \mathcal{C}(\sigma, (0)) () &\equiv \sigma(0()) \\ \mathcal{C}(\sigma, (1)) () &\equiv \sigma(1()) \end{aligned}$$

- ▶ Fonction 1 =  $\mathcal{C}(\sigma, (0))$   $\sigma(0()) = 1$
- ▶ Fonction 2 =  $\mathcal{C}(\sigma, (1))$   $\sigma(1()) = 2$

## Exemples

$f$  d'arité 0  $\rightarrow$  pas de  $g_i$  donc arité quelconque

$$\begin{aligned}\mathcal{C}(f, (g_1, \dots, g_p)) (n_1, \dots, n_q) &\equiv f(g_1(n_1, \dots, n_q), \dots, g_p(n_1, \dots, n_q)) \\ \mathcal{C}(0, ()) (n) &\equiv 0() \\ \mathcal{C}(0, ()) (n, m) &\equiv 0()\end{aligned}$$

► Fonction  $0^1(n) = \mathcal{C}(0, ())$  (pas de  $g_i$ )

$$0^1(n) = 0()$$

## Exemples

$f$  d'arité 0  $\rightarrow$  pas de  $g_i$  donc arité quelconque

$$\begin{aligned} \mathcal{C}(f, (g_1, \dots, g_p)) (n_1, \dots, n_q) &\equiv f(g_1(n_1, \dots, n_q), \dots, g_p(n_1, \dots, n_q)) \\ \mathcal{C}(0, ()) (n) &\equiv 0() \\ \mathcal{C}(0, ()) (n, m) &\equiv 0() \end{aligned}$$

- ▶ Fonction  $0^1(n) = \mathcal{C}(0, ())$  (pas de  $g_i$ )  $0^1(n) = 0()$
- ▶ Fonction  $1^1(n) = \mathcal{C}(\sigma, 0^1)$   $1^1(n) = \sigma(0()) = 1$

## Exemples

$f$  d'arité 0  $\rightarrow$  pas de  $g_i$  donc arité quelconque

$$\begin{aligned} \mathcal{C}(f, (g_1, \dots, g_p)) (n_1, \dots, n_q) &\equiv f(g_1(n_1, \dots, n_q), \dots, g_p(n_1, \dots, n_q)) \\ \mathcal{C}(0, ()) (n) &\equiv 0() \\ \mathcal{C}(0, ()) (n, m) &\equiv 0() \end{aligned}$$

- ▶ Fonction  $0^1(n) = \mathcal{C}(0, ())$  (pas de  $g_i$ )  $0^1(n) = 0()$
- ▶ Fonction  $1^1(n) = \mathcal{C}(\sigma, 0^1)$   $1^1(n) = \sigma(0()) = 1$
- ▶ Fonction  $2^1(n) = \mathcal{C}(\sigma, 1^1)$   $2^1(n) = \sigma(\sigma(0())) = 2$

## Exemples

$f$  d'arité 0  $\rightarrow$  pas de  $g_i$  donc arité quelconque

$$\begin{aligned} \mathcal{C}(f, (g_1, \dots, g_p)) \quad (n_1, \dots, n_q) &\equiv f(g_1(n_1, \dots, n_q), \dots, g_p(n_1, \dots, n_q)) \\ \mathcal{C}(0, ()) \quad (n) &\equiv 0() \\ \mathcal{C}(0, ()) \quad (n, m) &\equiv 0() \end{aligned}$$

- ▶ Fonction  $0^1(n) = \mathcal{C}(0, ())$  (pas de  $g_i$ )  $0^1(n) = 0()$
- ▶ Fonction  $1^1(n) = \mathcal{C}(\sigma, 0^1)$   $1^1(n) = \sigma(0()) = 1$
- ▶ Fonction  $2^1(n) = \mathcal{C}(\sigma, 1^1)$   $2^1(n) = \sigma(\sigma(0())) = 2$
- ▶ Fonction  $0^2(n, m) = \mathcal{C}(0, ())$  (pas de  $g_i$ )  $0^2(n, m) = 0()$

# Exemples

$$\begin{array}{l} \mathcal{C}(f, (g_1, \dots, g_p)) \quad (n_1, \dots, n_q) \equiv f(g_1(n_1, \dots, n_q), \dots, g_p(n_1, \dots, n_q)) \\ \mathcal{C}(\sigma, (\sigma)) \quad (n) \equiv \sigma(\sigma(n)) \end{array}$$

► Fonction  $+2(n) = \mathcal{C}(\sigma, \sigma)$

$$+2(n) = n + 2$$

## Opérateurs de récursion

- ▶  $f$  d'arité  $p$
- ▶  $g$  d'arité  $p + 2$ ,
- ▶ **Opérateur de récursion**  $\mathcal{R}(f, g)$  :

$$\mathcal{R}(f, g)(n_1, \dots, n_p, i) \equiv \begin{cases} f(n_1, \dots, n_p) & \text{si } i = 0 \\ g(n_1, \dots, n_p, m, \mathcal{R}(f, g)(n_1, \dots, n_p, m)) & \text{si } i = m + 1 \end{cases}$$

## Opérateurs de récursion

- ▶  $f$  d'arité  $p$
- ▶  $g$  d'arité  $p + 2$ ,
- ▶ **Opérateur de récursion**  $\mathcal{R}(f, g)$  :

$$\mathcal{R}(f, g)(n_1, \dots, n_p, i) \equiv \begin{cases} f(n_1, \dots, n_p) & \text{si } i = 0 \\ g(n_1, \dots, n_p, m, \mathcal{R}(f, g)(n_1, \dots, n_p, m)) & \text{si } i = m + 1 \end{cases}$$

- ▶ Autrement dit si  $\vec{n} = n_1, \dots, n_p$  :

$$\mathcal{R}(f, g) = h \text{ t.q. } \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m + 1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

## Exemple : +

$$\mathcal{R}(f, g) = h \text{ t.q. } \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m + 1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

► Addition :

$$\begin{aligned} +(n_1, 0) &= n_1 \\ +(n_1, n_2 + 1) &= 1 + (+(n_1, n_2)) \end{aligned}$$

## Exemple : +

$$\mathcal{R}(f, g) = h \text{ t.q. } \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m + 1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

► Addition :

$$\begin{aligned} +(n_1, 0) &= \pi_1^1(n_1) \\ +(n_1, n_2 + 1) &= 1 + (+(n_1, n_2)) \end{aligned}$$

## Exemple : +

$$\mathcal{R}(f, g) = h \text{ t.q. } \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m + 1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

► Addition :

$$\begin{aligned} +(n_1, 0) &= \pi_1^1(n_1) \\ +(n_1, n_2 + 1) &= \sigma(+(n_1, n_2)) \end{aligned}$$

## Exemple : +

$$\mathcal{R}(f, g) = h \text{ t.q. } \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m + 1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

► Addition :

$$\begin{aligned} +(n_1, 0) &= \pi_1^1(n_1) \\ +(n_1, n_2 + 1) &= \sigma(\pi_3^3(n_1, n_2, +(n_1, n_2))) \end{aligned}$$

## Exemple : +

$$\mathcal{R}(f, g) = h \text{ t.q. } \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m + 1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

► Addition :

$$\begin{aligned} +(n_1, 0) &= \pi_1^1(n_1) \\ +(n_1, n_2 + 1) &= \mathcal{C}(\sigma, (\pi_3^3))(n_1, n_2, +(n_1, n_2)) \end{aligned}$$

## Exemple : +

$$\mathcal{R}(f, g) = h \text{ t.q. } \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m + 1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

► Addition :

$$\begin{aligned} +(n_1, 0) &= \pi_1^1(n_1) \\ +(n_1, n_2 + 1) &= \mathcal{C}(\sigma, (\pi_3^3))(n_1, n_2, +(n_1, n_2)) \end{aligned}$$

► Fonction  $+$  =  $\mathcal{R}(\pi_1^1, (\mathcal{C}(\sigma, (\pi_3^3))))$

## Abus de notation

- ▶ Plus lisible : infixe + projections implicites :

$$n_1 + n_2 \equiv \begin{cases} n_1 + 0 & = n_1 \\ n_1 + (n_2 + 1) & = \sigma(n_1 + n_2) \end{cases}$$

- ▶ Attention aux abus

# Abus de notation

- ▶ Plus lisible : infixe + projections implicites :

$$n_1 + n_2 \equiv \begin{cases} n_1 + 0 & = n_1 & \in \mathcal{F}(n_1) \\ n_1 + (n_2 + 1) & = \sigma(n_1 + n_2) & \in \mathcal{F}(n_1, n_2, (n_1 + n_2)) \end{cases}$$

- ▶ Attention aux abus

Exercice :  $-$ ,  $\times$ 

$$\mathcal{R}(f, g) = h \text{ t.q. } \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m+1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

$$n_1 - n_2 \equiv \begin{cases} n_1 - 0 & = n_1 \\ n_1 - (n_2 + 1) & = (n_1 - n_2) - 1 \end{cases}$$

Exercice :  $-$ ,  $\times$ 

$$\mathcal{R}(f, g) = h \quad \text{t.q.} \quad \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m + 1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

$$n_1 - n_2 \equiv \begin{cases} n_1 - 0 & = n_1 \\ n_1 - (n_2 + 1) & = \text{pred}(n_1 - n_2) \end{cases}$$

Exercice :  $-$ ,  $\times$ 

$$\mathcal{R}(f, g) = h \text{ t.q. } \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m+1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

$$n_1 - n_2 \equiv \begin{cases} n_1 - 0 & = n_1 \\ n_1 - (n_2 + 1) & = \text{pred}(\pi_3^3(n_1, n_2, (n_1 - n_2))) \end{cases}$$

►  $\text{moins} = \mathcal{R}(\pi_1^1, \mathcal{C}(\text{pred}, (\pi_3^3)))$

Exercice :  $-$ ,  $\times$ 

$$\mathcal{R}(f, g) = h \text{ t.q. } \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m+1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

$$n_1 - n_2 \equiv \begin{cases} n_1 - 0 & = n_1 \\ n_1 - (n_2 + 1) & = \text{pred}(\pi_3^3(n_1, n_2, (n_1 - n_2))) \end{cases}$$

$$\text{prec}(m) \equiv \begin{cases} \text{pred}(0) & = 0 \\ \text{pred}(m+1) & = m \end{cases}$$

►  $\text{moins} = \mathcal{R}(\pi_1^1, \mathcal{C}(\text{pred}, (\pi_3^3)))$

Exercice :  $-$ ,  $\times$ 

$$\mathcal{R}(f, g) = h \text{ t.q. } \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m+1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

$$n_1 - n_2 \equiv \begin{cases} n_1 - 0 & = n_1 \\ n_1 - (n_2 + 1) & = \text{pred}(\pi_3^3(n_1, n_2, (n_1 - n_2))) \end{cases}$$

$$\text{prec}(m) \equiv \begin{cases} \text{pred}(0) & = 0 \\ \text{pred}(m+1) & = \pi_1^2(m, \text{pred}(m)) \end{cases}$$

►  $\text{moins} = \mathcal{R}(\pi_1^1, \mathcal{C}(\text{pred}, (\pi_3^3)))$

Exercice :  $-$ ,  $\times$ 

$$\mathcal{R}(f, g) = h \text{ t.q. } \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m+1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

$$n_1 - n_2 \equiv \begin{cases} n_1 - 0 & = n_1 \\ n_1 - (n_2 + 1) & = \text{pred}(\pi_3^3(n_1, n_2, (n_1 - n_2))) \end{cases}$$

$$\text{prec}(m) \equiv \begin{cases} \text{pred}(0) & = 0 \\ \text{pred}(m+1) & = \pi_1^2(m, \text{pred}(m)) \end{cases}$$

- ▶  $\text{prec} = \mathcal{R}(0, \pi_1^2)$
- ▶  $\text{moins} = \mathcal{R}(\pi_1^1, \mathcal{C}(\text{pred}, (\pi_3^3)))$

# Fonctions récursives primitives

Ensemble des fonctions récursives primitives défini comme **le plus petit ensemble**  $PR$  tel que :

- ▶ Les fonctions récursives de base  $(0, \sigma, \pi_i^k)$  sont dans  $PR$
- ▶  $PR$  est clos par composition
- ▶  $PR$  est clos par récursion primitive.

## Limite des fonction prim. réc.

- ▶  $\exists d$  fonction calculable pas primitive récursive (diagonalisation) :

$$d(n) \equiv f_n + 1 \text{ avec } \mathcal{F} = \{f_1, f_2 \dots\}$$

- ▶  $\forall n, d(n) \neq f_n(n)$
- ▶ Pourtant calculable facilement car  $\mathcal{F}$  réc. énum. :

```
d = begin  
    énumérer les  $f_i$  jusqu'à  $f_n$  ;  
    évaluer  $v = f_n(n)$  ;  
    retourner  $v + 1$   
end
```

# Minimisation

- ▶ Opérateur supplémentaire
- ▶ Permet la « récurrence vers le haut »

$$\mu i. q(n_1, \dots, n_p) \equiv \begin{cases} \text{Le plus petit } i \text{ t.q. } q(n_1, \dots, n_p, i) = 1 \\ 0 \text{ si il n'existe pas de tel } i \end{cases}$$

- ▶ Permet les énumérations non bornée (diagonalisation)

# Fonction récursive partielles

$$\mu i. q(n_1, \dots, n_p) \equiv \begin{cases} \text{Le plus petit } i \text{ t.q. } q(n_1, \dots, n_p, i) = 1 \\ 0 \text{ il n'existe pas de tel } i \end{cases}$$

- ▶  $i$  n'existe pas toujours → énumérations infinies
  - ▶ Fonctions non calculables
  - ▶  $\Rightarrow$  Fonction récursive *partielles*
- ▶ « il existe toujours de tel  $i$  »
  - ▶  $\Rightarrow$  Fonction récursive *totales* sinon



## Référence

Cours de JF Raskin(Univ. Libre Bruxelles

[www.ulb.ac.be/di/ssd/jfr/slides-calculabilite-complexite.pdf](http://www.ulb.ac.be/di/ssd/jfr/slides-calculabilite-complexite.pdf))

# Correspondance F.R. $\Rightarrow \lambda$ -calcul

**Théorème :**  $\forall f$  récursive partielle  $\Rightarrow \exists$  un  $\lambda$ -terme  $f_\lambda$  t.q. :

- ▶ Si  $f$  est définie en  $(n_1, \dots, n_k)$  alors  $f_\lambda \bar{n}_1 \dots \bar{n}_k$  a pour forme normale :  
 $\overline{f(n_1 \dots n_k)}$ .
- ▶ Sinon,  $f_\lambda \bar{n}_1 \dots \bar{n}_k$  n'a pas de forme normale.

Où  $\bar{n}$  désigne l'entier de Church correspondant à  $n$ .

## Fonctions de base

- ▶  $0 \rightsquigarrow \bar{0}$  (rappel :  $\bar{0} = \lambda f.\lambda y.y$ )
- ▶  $\sigma \rightsquigarrow SUCC$  ( $SUCC = \lambda n.\lambda f.\lambda x.f\ n\ f\ x$ )
- ▶  $\pi_i^k \rightsquigarrow \lambda x_1 \dots \lambda x_i \dots \lambda x_k.x_i$

## Composition

- ▶  $\mathcal{C}(X, (\Phi^1 \dots \Phi^m)) \rightsquigarrow \lambda \vec{x}. (X_\lambda (\Phi_\lambda^1 \vec{x}) \dots (\Phi_\lambda^m \vec{x}))$
- ▶ Ex :  $\mathcal{C}(\pi_1^2, (\sigma, \pi_1^1)) \rightsquigarrow \lambda x_1. ((\lambda x_2. \lambda x_3. x_2) (SUCC x_1) ((\lambda x. x) x_1))$

## Composition

- ▶  $\mathcal{C}(X, (\Phi^1 \dots \Phi^m)) \rightsquigarrow \lambda \vec{x}. (X_\lambda (\Phi_\lambda^1 \vec{x}) \dots (\Phi_\lambda^m \vec{x}))$
- ▶ Ex :  $\mathcal{C}(\pi_1^2, (\sigma, \pi_1^1)) \rightsquigarrow \lambda x_1. ((\lambda x_2. \lambda x_3. x_2) (SUCC\ x_1) ((\lambda x. x) x_1))$
- ▶ Petit problème : si  $\Phi^j$  ne termine pas,  $\mathcal{C}$  peut terminer quand même
- ▶ Astuce :  $Eval(f) = \lambda x. ((x (\lambda z. z)) f\ x)$  ( $x$  doit s'évaluer en  $\lambda y, \dots$ )
- ▶ Ex :  $\lambda x_1. (Eval(\bar{0})(\Phi_j)) = \lambda x_1. \lambda x. ((x (\lambda z. z)) \bar{0}\ x)$

# Récursion

$$\mathcal{R}(f, g) = h \text{ t.q. } \begin{cases} h(\vec{n}, 0) & \equiv f(\vec{n}) \\ h(\vec{n}, m+1) & \equiv g(\vec{n}, m, h(\vec{n}, m)) \end{cases}$$

- On cherche  $h_\lambda$  t.q.

$$h_\lambda \equiv_\beta \lambda \vec{n}. \lambda m.$$

IFTHENELSE (ISZERO  $m$ )

( $f_\lambda \vec{n}$ )

( $g_\lambda \vec{n}$  (PRED  $m$ ) ( $h_\lambda \vec{n}$  (PRED  $m$ )))

# Récursion

- ▶ On cherche  $h_\lambda$  t.q.

$$h_\lambda \equiv_\beta \lambda \vec{n}. \lambda m. \\ \text{IFTHENELSE (ISZERO } m) \\ (f_\lambda \vec{n}) \\ (g_\lambda \vec{n} (\text{PRED } m) (h_\lambda \vec{n} (\text{PRED } m))))$$

# Récursion

- ▶ On cherche  $h_\lambda$  t.q.

$$\begin{aligned}
 h_\lambda \equiv_\beta & \quad \lambda \vec{n}. \lambda m. \\
 & \quad \text{IFTHENELSE (ISZERO } m) \\
 & \quad \quad (f_\lambda \vec{n}) \\
 & \quad \quad (g_\lambda \vec{n} (\text{PRED } m) (h_\lambda \vec{n} (\text{PRED } m))))
 \end{aligned}$$

# Récursion

- On cherche  $h_\lambda$  t.q.

$$\begin{array}{c}
 h_\lambda \equiv_\beta (\lambda h. \lambda \vec{n}. \lambda m. \\
 \quad \text{IFTHENELSE (ISZERO } m) \\
 \quad \quad (f_\lambda \vec{n}) \\
 \quad \quad (g_\lambda \vec{n} (\text{PRED } m) (h \vec{n} (\text{PRED } m)))) h_\lambda \\
 \hline
 h_\lambda \equiv_\beta (\lambda h. t_h) h_\lambda
 \end{array}$$

# Récursion

- ▶ On cherche  $h_\lambda$  t.q.

$$\begin{array}{c}
 h_\lambda \equiv_\beta (\lambda h. \lambda \vec{n}. \lambda m. \\
 \quad \text{IFTHENELSE (ISZERO } m) \\
 \quad \quad (f_\lambda \vec{n}) \\
 \quad \quad (g_\lambda \vec{n} (\text{PRED } m) (h \vec{n} (\text{PRED } m)))) h_\lambda \\
 \hline
 h_\lambda \equiv_\beta (\lambda h. t_h) h_\lambda
 \end{array}$$

- ▶ Opérateur de point fixe  $\mathbf{Y}$  t.q. :  $\mathbf{Y} t \equiv_\beta t (\mathbf{Y} t)$
- ▶ Solution  $\boxed{h_\lambda = \mathbf{Y} (\lambda h. t_h)} \equiv_\beta (\lambda h. t_h) (\mathbf{Y} (\lambda h. t_h)) = (\lambda h. t_h) h_\lambda$ . OK.

# Minimisation

- ▶ On veut définir :

$$\mu_i.q(n_1, \dots, n_p) \equiv \begin{cases} \text{Le plus petit } i \text{ t.q. } q(n_1, \dots, n_p, i) = 1 \\ 0 \text{ si il n'existe pas de tel } i \end{cases}$$

- ▶ Récursivement en partant de zéro :  $\mu_i.q(n_1, \dots, n_p) = h(0)$  avec :

$$\begin{cases} h(i) \equiv i \text{ si } q(n_1, \dots, n_p, i) = 1 \\ h(i + 1) \text{ sinon} \end{cases}$$

- ▶ Opérateur de point fixe

# Minimisation

$$\begin{cases} h(i) \equiv i \text{ si } q(n_1, \dots, n_p, i) = 1 \\ h(i + 1) \text{ sinon} \end{cases}$$

- On cherche  $h_\lambda$  t.q.

$$h_\lambda \equiv_\beta \lambda \vec{n}. \lambda m. \\ \text{IFTHENELSE (ISZERO (q } \vec{n} \text{ m))} \\ (h_\lambda \vec{n} (\text{SUCC } m)) \\ m$$

# Minimisation

$$\begin{cases} h(i) \equiv i \text{ si } q(n_1, \dots, n_p, i) = 1 \\ h(i + 1) \text{ sinon} \end{cases}$$

- On cherche  $h_\lambda$  t.q.

$$h_\lambda \equiv_\beta \lambda \vec{n}. \lambda m. \\ \text{IFTHENELSE (ISZERO (q } \vec{n} \text{ m))} \\ (h_\lambda \vec{n} (\text{SUCC } m)) \\ m$$

# Minimisation

$$\begin{cases} h(i) \equiv i \text{ si } q(n_1, \dots, n_p, i) = 1 \\ h(i + 1) \text{ sinon} \end{cases}$$

- On cherche  $h_\lambda$  t.q.

$$h_\lambda \equiv_\beta (\lambda h. \lambda \vec{n}. \lambda m. \text{IFTHENELSE} (\text{ISZERO} (q \vec{n} m)) (h \vec{n} (\text{SUCC } m))) h_\lambda$$

$$\underbrace{\hspace{15em}}_{h_\lambda \equiv_\beta (\lambda h. u_h) h_\lambda}$$

# Minimisation

$$\begin{cases} h(i) \equiv i \text{ si } q(n_1, \dots, n_p, i) = 1 \\ h(i + 1) \text{ sinon} \end{cases}$$

- ▶ On cherche  $h_\lambda$  t.q.

$$\underbrace{h_\lambda \equiv_\beta (\lambda h. \lambda \vec{n}. \lambda m. \text{IFTHENELSE} (\text{ISZERO} (q \vec{n} m)) (h \vec{n} (\text{SUCC } m))) h_\lambda}_m$$

$$h_\lambda \equiv_\beta (\lambda h. u_h) h_\lambda$$

- ▶ Opérateur de point fixe  $\mathbf{Y}$  t.q. :  $\mathbf{Y} t \equiv_\beta t (\mathbf{Y} t)$

- ▶ Solution  $\boxed{h_\lambda = \mathbf{Y} (\lambda h. u_h)}$

## Exercice

- ▶ Définir une fonction récursive  $SUP_2$  t.q.  $SUP_2(n) = 1$  ssi  $n > 2$ .
- ▶ Définir un  $\lambda$ -terme  $SUP_{2\lambda}$  t.q.  $(SUP_{2\lambda} \bar{n}) \rightarrow_{\beta}^* TRUE$  ssi  $\bar{n} > 2$ .
- ▶ Définir un  $\lambda$ -terme correspondant à  $\mu i.(\lambda i.(SUP_2 (SUCC i)))$
- ▶ Effectuer le calcul en stratégie normale



## Rappel définition M.T.

$$M = (Q, \Sigma, q_0, \delta, F)$$

- ▶  $Q$  ensemble (fini) d'états,  $F \subset Q$  états finaux,  $q_0 \in Q$  état initial
- ▶  $\Sigma$  Alphabet (fini) du ruban
- ▶  $\delta$  programme (ensemble d'instructions)
- ▶ Instruction  $i$  :  $(Q \times \Sigma) \times (Q \times \Sigma \times \{\leftarrow, \rightarrow, -\})$   
((état, symb. lu), (état, symb. écrit, déplacement))
- ▶ Configuration  $(q, k, w)$  : état courant  $q \in Q$ ,  $w \in \Sigma^*$  contenu ruban courant,  $k \in \mathbb{N}$  position curseur (ruban infini à droite seulement)
- ▶ Transition :  $(q, k, w) \mapsto (q', k', w')$ .

# Codage

- ▶ Fonctions récursive : seulement les entiers.
- ▶ **Lemme** : Sur alphabet fini  $\Sigma$ , coder le ruban par des entiers naturels (Gödelisation).
- ▶ **Preuve** :
  - ▶ Supposons  $|\Sigma| = k$
  - ▶ Codage  $gd$  de  $\Sigma = \{s_1, s_2, \dots, s_k\}$  :
    - ▶  $gd : \Sigma \rightarrow \{1, \dots, k\}$
    - ▶  $gd : s_i \mapsto i$
  - ▶ Codage d'une chaîne  $w = a_1 \dots a_l$  comme un entier en base  $k$  :

$$gd(w) = \sum_{i=0}^l k^{(l-i)} gd(a_i)$$

Ex :  $s_2 s_4 \mapsto k^2 \cdot 2 + k^1 \cdot 4$

- ▶ Important : Codage injectif et *lui-même calculable*.

## Extension du codage au configuration d'un M.T.

- ▶ Même codage mais sur un alphabet plus grand :
- ▶  $\sigma' = \Sigma \cup \{q_0, \dots, q_m\}$
- ▶  $gd(q, k, w) = gd(w_1 \dots w_{k-1} q w_k \dots w_n)$   
( $k$  : position curseur dans le ruban).

# Énoncé du théorème

- ▶ **Théorème** : Soit  $M$  une M.T. calculant la fonction  $f_M : \Sigma^* \rightarrow \Sigma^*$ , il existe une fonction  $\mu$ -calculable  $f$  telle que :

$$\forall w \in \Sigma^*, gd(f_M(q_0, 0, w)) = f(gd(q_0, 0, w)).$$

- ▶ Remarque : **une** machine  $\rightsquigarrow$  **une** fonction récursive.

## Fonctions récursives nécessaires

- ▶  $f =$  composition ( $C(f,g)$ ) sur fonctions récursives plus basiques.
- ▶ Arguments et résultats : entiers codant ruban ou configuration

## Fonctions récursives nécessaires

- ▶  $init : \begin{cases} \mathbb{N} \rightarrow \mathbb{N} \\ gd(w) \mapsto \text{config. initiale de } M \text{ sur } w \\ gd(w) \mapsto gd(q_0, 0, w) \end{cases}$
- ▶ Récursive :  
 $gd(q_0, 0, w) \equiv gd(q_0 w) \equiv |\Sigma|^{|w|} \cdot gd(w) \equiv \text{simple multiplication.}$

▶ *ConfigSuiv* :

$$\begin{cases} \mathbb{N} \rightarrow \mathbb{N} \\ gd(q, k, w) \mapsto \text{config. suivante de } M \text{ depuis config. } w \\ gd(q, k, w) \mapsto gd(q', k', w') \end{cases}$$

▶ Récursive :

- ▶ Extraire curseur  $q$  (divisions par puissances de  $|\sigma|$ )
- ▶ Extraire symbole  $s$  qui suit  $q$  (re-division)
- ▶ Appliquer transition correspondant à  $s$  et  $q$
- ▶ Reconstruire préfixe du ruban (multiplication)

► *NStep* :

$$\left\{ \begin{array}{l} \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\ (gd(q, k, w), n) \mapsto \text{config. après } n \text{ étape de } M \text{ depuis config. } w \\ (gd(q, k, w), n) \mapsto gd(q', k', w') \end{array} \right.$$

► Récursive : Opérateur de récursion :

$$\left\{ \begin{array}{l} NStep(x, 0) = x \\ NStep(x, n + 1) = ConfigSuiv(NStep(x, n)) \end{array} \right.$$

►  $Stop : \begin{cases} \mathbb{N} \rightarrow \mathbb{N} \\ (gd(q, k, w)) \mapsto \begin{cases} 1 \text{ si } q \text{ final} \\ 0 \text{ sinon} \end{cases} \end{cases}$

► Récursive : test  $q \in Q_{\text{finaux}}$

- ▶ *Sortie* : 
$$\left\{ \begin{array}{l} \mathbb{N} \rightarrow \mathbb{N} \\ gd(w) \mapsto \text{contenu du ruban dans config. (finale) } w \\ gd(q_0, 0, w) \mapsto gd(w) \end{array} \right.$$
- ▶ *Réursive* : enlève juste  $q$  dans  $w$  (division).

$$f(x) = \text{Sortie}(\text{NStep}(\text{init}(x), \text{NbDePas}(x)))$$

$$\text{où : } \text{NbDePas}(x) = \mu_i. \text{Stop}(\text{NStep}(\text{Init}(x), i))$$