# MODELISATION LOGICO-COMBINATOIRE DES REALITES INFORMATIQUES

## B. ROBINET

Patrick Bellot
Télécom Paris

- Dans cette section, nous montrons comment réaliser les entités informatiques classiques :
  - les booléens, leurs opérations et la conditionnelles ;
  - les entiers naturels et leurs opérations ;
  - les listes.

- Nous voyons aussi comment transformer une définition implicite (équation récursive) ou une définition explicite de fonction en un terme calculant la fonction.

# Les booléens (G. Boole, 1849)

- En logique, on utilise deux marques, t et f, pour représenter la vérité et la fausseté des propositions.

- On choisit de modéliser les deux valeurs de vérité par :

  $\underline{t} \equiv_{def} K$

  $\underline{f} \equiv_{def} K\ I$

- $\underline{t}$ et $\underline{f}$ sont les modèles de t et f.

# Les booléens : expression conditionnelle

- On a :

  $\underline{t}\ X\ Y \equiv K\ X\ Y := X$

  $\underline{f}\ X\ Y \equiv K\ I\ X\ Y := I\ Y := Y$

- On peut donc modéliser l'expression conditionnelle

  *if* P *then* F *else* G

  par :

  P F G

# Les booléens : la conjonction

- On a (vieux Lispien) : $x \wedge y = \textit{if}\ x\ \textit{then}\ y\ \textit{else}\ \underline{f}$
- Donc on cherche $\underline{\wedge}$ tel que $\underline{\wedge}\ x\ y = x\ y\ \underline{f}$
- Cherchons :

$\underline{\wedge}\ x\ y$
$= x\ y\ \underline{f}$
$= C\ x\ \underline{f}\ y$ *permutation*
$= C\ C\ \underline{f}\ x\ y$ *permutation*

- On peut donc prendre :

$$\underline{\wedge} \equiv_{def} C\ C\ \underline{f}$$

# Les booléens : la disjonction

- On a (vieux Lispien) : $x \vee y = \mathit{if}\ x\ \mathit{then}\ \underline{t}\ \mathit{else}\ y$
- Donc on cherche $\underline{\vee}$ tel que $\underline{\vee}\ x\ y = x\ \underline{t}\ y$
- Cherchons :

$$\underline{\vee}\ x\ y$$
$$= x\ \underline{t}\ y$$
$$= C_*\ \underline{t}\ x\ y$$

- On peut donc prendre :

$$\underline{\vee} \equiv_{\text{def}} C_*\ \underline{t}$$

# Les booléens : la négation

- On a : $\neg\, x = \mathit{if}\ x\ \mathit{then}\ \underline{f}\ \mathit{else}\ \underline{t}$
- Donc on cherche $\underline{\neg}$ tel que $\underline{\neg}\, x = x\ \underline{f}\ \underline{t}$
- Cherchons :

$$\underline{\neg}\, x = x\ \underline{f}\ \underline{t}$$
$$= C_*\ \underline{f}\ x\ \underline{t}$$
$$= C\ (C_*\ \underline{f})\ \underline{t}\ x$$

- On peut donc prendre :

$$\underline{\neg} \equiv_{def} C\ (C_*\ \underline{f})\ \underline{t}$$

# Les entiers naturels

- Nous allons développer un modèle des entiers naturels appelé *entiers* ou *itérateurs* de CHURCH.

- Nous introduisons la notation $f^n\ x \equiv_{def} \underbrace{f\ (f\ \ldots\ (f}_{n \times f}\ x)..)$

  définie par :

$$\begin{cases} f^0\ x \equiv_{def} x \\ f^{n+1}\ x \equiv_{def} f\ (f^n\ x) \end{cases}$$

- On a bien entendu : $f^{p+q}\ x \equiv f^p\ (f^q\ x)$. Exercice…

# Les entiers naturels de CHURCH

- On pose :

$$\begin{cases} \underline{0} \equiv_{def} \mathrm{K\ I} & (\textit{zéro}) \\ \underline{\mathrm{s}} \equiv_{def} \mathrm{S\ B} & (\textit{successeur}) \end{cases}$$

- Le modèle de l'entier naturel n est $\underline{n} \equiv_{def} \underline{\mathrm{s}}^{n}\ \underline{0}$

- De tels modèles d'entiers sont appelés des *numéraux*.

# Les itérateurs de CHURCH

- Théorème. Soit $\underline{n}$ un entier de CHURCH, alors on a

  $\underline{n}\ f\ x\ :=\ f^n\ x$

- L'entier de CHURCH $\underline{n}$ permet donc d'itérer une fonction $f$ sur un argument $x$.

- Démonstration. Par récurrence.

  Si $n=0$,

  $\underline{n}\ f\ x \equiv \underline{0}\ f\ x \equiv K\ I\ f\ x := I\ x := x \equiv f^0\ x$

  Si $n=k+1$,

  $\underline{k+1}\ f\ x \equiv \underline{s}\ \underline{k}\ f\ x \equiv S\ B\ \underline{k}\ f\ x := B\ f\ (\underline{k}\ f)\ x = f\ (\underline{k}\ f\ x) = f\ (f^k\ x) = f^{\,k+1}\ x$

# Les entiers naturels : test d'égalité à zéro

- On recherche z tel que :

$$\begin{cases} z\ 0 = t \\ z\ n{+}1 = f \end{cases}$$

- Dans un cas comme cela, on cherche à examiner l'argument de la fonction. Il n'existe pas de terme permettant d'examiner le « contenu » d'un autre terme, par exemple savoir si c'est une application ou pas. On sait même que c'est IMPOSSIBLE…
- Le seul moyen d'« examiner » un terme est de l'appliquer à des arguments et de regarder ce que cela donne.
- On cherchera donc z sous la forme z n = n U en recherchant U pour obtenir le résultat voulu. Ca ne marchera pas, on cherchera z sous la forme z n = n U V en recherchant U et V.

- On cherche $\underline{z}$ sous la forme $\underline{z}\ n = n\ U\ V$, il faut donc trouver $U$ et $V$ tels que :

$$\begin{cases} \underline{0}\ U\ V = \underline{t} \\ \underline{n+1}\ U\ V = \underline{f} \end{cases}$$

- $\underline{0}\ U\ V \equiv K\ I\ U\ V = I\ V = V$, il faut donc $V \equiv \underline{t}$
- $\underline{n+1}\ U\ V \equiv \underline{s}\ \underline{n}\ U\ V \equiv S\ B\ \underline{n}\ U\ V = B\ U\ (\underline{n}\ U)\ V = U\ (\underline{n}\ U\ V) = \underline{f}$, on peut obtenir ce résultat en prenant $U \equiv K\ \underline{f}$
- On veut donc :

  $\underline{z}\ n = n\ (K\ \underline{f})\ \underline{t} = C_*\ (K\ \underline{f})\ n\ \underline{t} = C\ (C_*\ (K\ \underline{f}))\ \underline{t}\ n$

- On prend donc : $\underline{z} \equiv_{def} C\ (C_*\ (K\ \underline{f}))\ \underline{t}$

# Les entiers naturels : l'addition

- On cherche $\underline{+}$ tel que $\underline{+}\ \underline{m}\ \underline{n} = \underline{m+n}$
- On a :

  $\underline{m+n} \equiv \underline{s}^{m+n}\ \underline{0} \equiv \underline{s}^{m}\ (\underline{s}^{n}\ \underline{0}) = \underline{m}\ \underline{s}\ (\underline{s}^{n}\ \underline{0}) \equiv \underline{m}\ \underline{s}\ \underline{n}$
- On veut donc :

  $\underline{+}\ \underline{m}\ \underline{n} = \underline{m}\ \underline{s}\ \underline{n} = C_{*}\ \underline{s}\ \underline{m}\ \underline{n}$

- On prend donc :

$$\underline{+} \equiv_{def} C_{*}\ \underline{s}$$

# Les entiers naturels : la multiplication

- On cherche $\underline{*}$ tel que $\underline{*}\ \underline{m}\ \underline{n} = \underline{m*n}$
- On a :

$$\underline{m*n} \equiv \underline{s}^{m*n}\ \underline{0} \equiv \underline{s}^{n}\ (\underline{s}^{n}\ (\ldots(\underline{s}^{n}\ \underline{0})..))$$

$$= \underline{n}\ \underline{s}\ (\underline{n}\ \underline{s}\ (\ \ldots\ (\underline{n}\ \underline{s}\ \underline{0})..)) = \underline{m}\ (\underline{n}\ \underline{s})\ \underline{0}$$

- On veut donc :

$$\underline{*}\ \underline{m}\ \underline{n} = \underline{m}\ (\underline{n}\ \underline{s})\ \underline{0} = C\ \underline{m}\ \underline{0}\ (\underline{n}\ \underline{s}) = C\ C\ \underline{0}\ \underline{m}\ (\underline{n}\ \underline{s})$$

$$= B\ (C\ C\ \underline{0}\ \underline{m})\ \underline{n}\ \underline{s} = C\ (B\ (C\ C\ \underline{0}\ \underline{m}))\ \underline{s}\ \underline{n}$$

$$= C\ C\ \underline{s}\ (B\ (C\ C\ \underline{0}\ \underline{m}))\ \underline{n} = B\ (C\ C\ \underline{s})\ B\ (C\ C\ \underline{0}\ \underline{m})\ \underline{n}$$

$$= B\ (B\ (C\ C\ \underline{s})\ B)\ (C\ C\ \underline{0})\ \underline{m}\ \underline{n}$$

- On peut donc prendre :

$$\underline{*} \equiv_{def} B\ (B\ (C\ C\ \underline{s})\ B)\ (C\ C\ \underline{0})$$

# Les entiers naturels

La soustraction

et

la division

seront vues plus tard.

# Les définitions explicites

- Nous avons eu affaire à des définitions explicites du style f x = S (x (± x 0)), à charge pour nous de déterminer f :

f x = S (x (± x 0))
= S (x (C ± 0 x))
= S (I x (C ± 0 x))
= S (S I (C ± 0 ) x)
= B S (S I (C ± 0 )) x

f x = S (x (± x 0))
= B S x (± x 0)
= B S x (C ± 0 x)
= S (B S) (C ± 0 ) x

- En agissant ainsi, on réalise l'ABSTRACTION de la variable x dans le terme S (x (± x 0)).
- Il existe en général plusieurs manières de réaliser une abstraction. Il n'y a AUCUNE raison pour que les résultats soient égaux…
- Si je trouve f, B f I convient aussi…

# Les définitions explicites

- Convenons de NOTER $(\lambda^* x.M)$ tout terme résultat d'une abstraction de $x$ dans $M$. C'est-à-dire un terme tel que :

  $(\lambda^* x.M)\ x := M$ et $x \notin (\lambda^* x.M)$

  et plus exactement tel que :

  $(\lambda^* x.M)\ N := [N / x]\ M$ et $x \notin (\lambda^* x.M)$

  où $[N / x]\ M$ DESIGNE le résultat de la substitution de $N$ à toutes les occurrences de $x$.
- Le processus de calcul d'un $(\lambda^* x.M)$ peut-il être automatisé ? La réponse est OUI !

# La substitution

- Si M et N sont des termes et x une variable, la substitution de N à x dans M, notée [N / x]M se définit par induction :
  - $[N / x]\, c \equiv_{def} c$ si c est une constante (BASE)
  - $[N / x]\, x \equiv_{def} N$ (BASE)
  - $[N / x]\, y \equiv_{def} y$ si y est une variable et $y \neq x$ (BASE)
  - $[N / x]\, (P\ Q) \equiv_{def} ([N / x]P\ [N\ / x]Q)$ (PAS)
- [N / x]M désigne simplement le remplacement de TOUTES les occurrences de x dans M par N. Cela ne pose aucun problème car il n'y a pas de quantificateurs.

# La substitution

- Pourquoi définir la substitution ? Pour pouvoir faire des démonstrations par induction concernant la substitution.

- [N / x] M sera lu :
  - substitution de N à x dans M
  - ou remplacement de x par N dans M
  - ou N à la place de x dans M
  - ou M dans lequel on remplace x par N
  - etc.

# Exemple de substitution rigoureuse

$[A / x]\ (S\ (x\ (\underline{+}\ x\ \underline{0})))$

$\equiv [A / x]S\ [A / x](x\ (\underline{+}\ x\ \underline{0}))$

$\equiv S\ ([A / x]x\ [A / x](\underline{+}\ x\ \underline{0}))$

$\equiv S\ (A\ ([A / x](\underline{+}\ x)\ [A /x]\underline{0}))$

$\equiv S\ (A\ ([A / x]\underline{+}\ [A /x]x\ \underline{0}))$

$\equiv S\ (A\ (\underline{+}\ A\ \underline{0}))$

# Un algorithme d'abstraction $(\lambda^+x.M)$

- Soit x une variable et M un terme, on définit $(\lambda^+x.M)$ par induction sur M :
  - $(\lambda^+x.c) \equiv_{def} K\ c$ si c est une constante (BASE)
  - $(\lambda^+x.x) \equiv_{def} I$ (BASE)
  - $(\lambda^+x.y) \equiv_{def} K\ y$ si y est une variable et $y \neq x$ (BASE)
  - $(\lambda^+x.PQ) \equiv_{def} S\ (\lambda^+x.P)\ (\lambda^+x.Q)$

- Theorème. On a :

  $(\lambda^+x.M)\ N := [N / x]\ M$ et $x \notin (\lambda^+x.M)$

  et : $(\lambda^+x.M)$ est une forme normale

# Démonstration : $x \notin (\lambda^{+}x.M)$

*BASE*

- Si $M$ est une constante, $(\lambda^{+}x.M) \equiv K\ M$ ne contient pas $x$.
- Si $M \equiv x$, $(\lambda^{+}x.M) \equiv I$ ne contient pas $x$.
- Si $M \equiv y \neq x$, $(\lambda^{+}x.M) \equiv K\ y$ ne contient pas $x$.

*PAS*

- Si $M \equiv P\ Q$, $(\lambda^{+}x.M) \equiv S\ (\lambda^{+}x.P)\ (\lambda^{+}x.Q)$ ne contient pas $x$ puisque $(\lambda^{+}x.P)$ et $(\lambda^{+}x.Q)$ ne contiennent pas $x$ par hypothèse d'induction.

# Démonstration
# $(\lambda^{+}x.M)$ est une forme normale

*BASE*

- Si $M$ est une constante, $(\lambda^{+}x.M) \equiv K\ M$ est une f.n.
- Si $M \equiv x$, $(\lambda^{+}x.M) \equiv I$ est une f.n.
- Si $M \equiv y \neq x$, $(\lambda^{+}x.M) \equiv K\ y$ est une f.n.

*PAS*

- Si $M \equiv P\ Q$, $(\lambda^{+}x.M) \equiv S\ (\lambda^{+}x.P)\ (\lambda^{+}x.Q)$ est une f.n. puisque $(\lambda^{+}x.P)$ et $(\lambda^{+}x.Q)$ sont des f.n. par hypothèse d'induction.

# Démonstration : $(\lambda^{+}x.M)\ N := [N / x]\ M$

*BASE*

- Si M est une constante, $(\lambda^{+}x.M)\ N \equiv K\ M\ N := [N / x]\ M \equiv M$
- Si $M \equiv x$, $(\lambda^{+}x.M)\ N \equiv I\ N := N \equiv [N/ x]x \equiv [N/ x]M$
- Si $M \equiv y \neq x$, $(\lambda^{+}x.M)\ N \equiv K\ y\ N := y \equiv [N / x]\ y \equiv [N / x]\ M$

*PAS*

- Si $M \equiv P\ Q$,
  $(\lambda^{+}x.M)\ N \equiv S\ (\lambda^{+}x.P)\ (\lambda^{+}x.Q)\ N := (\lambda^{+}x.P)\ N\ ((\lambda^{+}x.Q)\ N)$
  Par hypothèse d'induction on a $(\lambda^{+}x.P)\ N := [N / x]P$ et
  $(\lambda^{+}x.Q)\ N := [N / x]Q$, donc :
  $(\lambda^{+}x.M)\ N := ([N / x]P)([N / x]Q) \equiv [N / x](P\ Q) \equiv [N / x]\ M$

# Exemple d'abstraction

$$\begin{aligned}
&\lambda^{+}x\,.\,S\ (x\ (\underline{+}\ x\ \underline{0}))\\
&\quad\equiv S\ (\lambda^{+}x\,.\,S)\ (\lambda^{+}x\,.\,(x\ (\underline{+}\ x\ \underline{0})))\\
&\quad\equiv S\ (K\ S)\ (S\ (\lambda^{+}x\,.\,x)\ (\lambda^{+}x\,.\,(\underline{+}\ x\ \underline{0})))\\
&\quad\equiv S\ (K\ S)\ (S\ I\ (S\ (\lambda^{+}x\,.\,\underline{+}\ x)\ (\lambda^{+}x\,.\,\underline{0})))\\
&\quad\equiv S\ (K\ S)\ (S\ I\ (S\ (S\ (\lambda^{+}x\,.\,\underline{+})\ (\lambda^{+}x\,.\,x))\ (K\ \underline{0})))\\
&\quad\equiv S\ (K\ S)\ (S\ I\ (S\ (S\ (K\ \underline{+})\ \Gamma)\ (K\ \underline{0})))
\end{aligned}$$

- On a pris $(\lambda^{+}x\,.\,\underline{+}) \equiv K\ \underline{+}$ et $(\lambda^{+}x\,.\,\underline{0}) \equiv K\ \underline{0}$ comme si $\underline{+}$ et $\underline{0}$ étaient atomiques…

# Définition explicite

- L'algorithme d'abstraction $\lambda^+$ permet à partir d'une définition explicite $f\ x = E(x)$ d'une fonction $f$ d'obtenir un terme $(\lambda^+ x\ .\ E(x))$ qui est un algorithme pour la fonction $f$.
- La notation $(\lambda^+ x\ .\ M)$ désigne le terme obtenu comme résultat de l'abstraction de $x$ dans $M$ par l'algorithme $\lambda^+$.
- On voit donc que deux combinateurs suffisent à transformer toute définition explicite en un terme de la théorie des combinateurs.

# Définition implicite

- On avait appelé *définition implicite* une équation récursive de la forme $f\ x = E(f,x)$.
- On commence par écrire $f = (\lambda^+x\ .\ E(f,x))$. C'est une équation en $f$ où la variable $x$ n'apparaît plus.
- Puis on écrit $f = (\lambda^+f\ .\ (\lambda^+x\ .\ E(f,x)))\ f$. En effet, $(\lambda^+f.(\lambda^+x.E(f,x)))\ f := [f/f\ ](\lambda^+x.E(f,x)) \equiv (\lambda^+x.E(f,x))$.
- De plus, $(\lambda^+f\ .\ (\lambda^+x\ .\ E(f,x)))$ est un terme ne contenant que des $S$ et des $K$. Il est appelé la *fonctionnelle associée à la définition récursive*. L'équation dit que $f$ est *un point-fixe de cette fonctionnelle*.

- De la *définition implicite* f x = E(f,x), on est arrivé à l'équation $f = (\lambda^{+} f . (\lambda^{+} x . E(f,x)))\ f$ qui est une équation de point-fixe.
- Une solution de cette équation nous est donnée par le combinateur de point-fixe Y :

$$f = Y\ (\lambda^{+} f . (\lambda^{+} x . E(f,x)))$$

- Les combinateurs S et K nous permettent donc de trouver un algorithme pour calculer l'une des solutions de toute équation récursive.
- [*Si mes souvenirs sont bons*] Y permet de calculer le plus petit point-fixe, c'est-à-dire la fonction la moins définie solution de l'équation récursive.

# Substitution simultanée

- La notation $[X/x, Y/y, \ldots, Z/z]M$ désigne le résultat de la *substitution simultanée* de $X$ à $x$, $Y$ à $y$, … et $Z$ à $z$ dans $M$.

- Exemple. $[y / x, z / y]\ (+\ x\ y) \equiv (+\ y\ z)$

- *Exercice. Donner une définition par induction de la substitution simultanée.*

# Abstraction multiple

- On pose

  $(\lambda^+ x, y, \ldots, z \,.\, M) \equiv_{def} (\lambda^+ x \,.\, (\lambda^+ y \,.\, (\ldots (\lambda^+ z \,.\, M)..)))$

- Théorème. On a les propriétés suivantes :
  - $(\lambda^+ x, y, \ldots, z \,.\, M)$ est une forme normale
  - $x \notin (\lambda^+ x, y, \ldots, z \,.\, M)$
  - $y \notin (\lambda^+ x, y, \ldots, z \,.\, M)$
  - …
  - $z \notin (\lambda^+ x, y, \ldots, z \,.\, M)$
  - $(\lambda^+ x, y, \ldots, z \,.\, M)\ X\ Y\ \ldots\ Z := [X/x, Y/y, \ldots, Z/z]M$
- *Démonstration. A faire en exercice.*

# Un peu de complexité (un tout petit peu… faut pas pousser !)

- On peut représenter un terme sous la forme d'un arbre binaire dont les nœuds sont des applications et dont les feuilles sont les atomes.
- On peut *mesurer* la taille d'un arbre comme étant le nombre de nœuds de cet arbre.
- L'équation $(\lambda^{+}x.PQ) \equiv_{def} S\ (\lambda^{+}x.P)\ (\lambda^{+}x.Q)$ nous dit que la taille de l'arbre résultat est au moins le double de la taille de $M$.
- Si on effectue $n$ abstractions consécutives, la taille de l'arbre est multipliée par $2^n$. Peut-on optimiser ?

# Une première amélioration

- Soit F tel que $x \notin F$ on remarque que $(\lambda^{+}x \,.\, F\ x) \neq F$ mais est un terme plus compliqué.
- Exemple :
  $(\lambda^{+}x \,.\, I\ x) \equiv S\ (\lambda^{+}x \,.\, I)\ (\lambda^{+}x \,.\, X) \equiv S\ (K\ I)\ I$
- Or F pourrait convenir puisque l'on demande à $(\lambda^{+}x \,.\, F\ x)$ de ne pas contenir x et d'être tel que $(\lambda^{+}x \,.\, F\ x)\ N := [N / x](F\ x) \equiv F\ N$
- On aurait alors :
  $(\lambda^{+}x \,.\, I\ x) \equiv I$

# Un algorithme d'abstraction $(\lambda^{++}x.M)$

- Soit x une variable et M un terme, on définit $(\lambda^{++}x.M)$ par induction sur M :
  - $(\lambda^{++}x.F\ x) \equiv_{def} F$ si $x \notin F$ (PRIORITAIRE)
  - $(\lambda^{++}x.c) \equiv_{def} K\ c$ si c est une constante (BASE)
  - $(\lambda^{++}x.x) \equiv_{def} I$ (BASE)
  - $(\lambda^{++}x.y) \equiv_{def} K\ y$ si y est une variable et $y \neq x$ (BASE)
  - $(\lambda^{++}x.PQ) \equiv_{def} S\ (\lambda^{++}x.P)\ (\lambda^{++}x.Q)$

- Théorème. On a :

  $(\lambda^{++}x.M)\ N := [N / x]\ M$ et $x \notin (\lambda^{++}x.M)$

- On perd la propriété de la forme normale.

# Une deuxième amélioration

- Soit F non atomique tel que $x \notin F$ on remarque que $(\lambda^{+}x \,.\, F) \neq K\ F$ mais est un terme plus compliqué.
- Exemple :

  $(\lambda^{+}x \,.\, I\ I) \equiv S\ (\lambda^{+}x \,.\, I)\ (\lambda^{+}x \,.\, I) \equiv S\ (K\ I)\ (K\ I)$
- Or K F pourrait convenir puisque l'on demande à $(\lambda^{+}x \,.\, F)$ de ne pas contenir x et d'être tel que $(\lambda^{+}x \,.\, F)\ N := [N / x]F \equiv F =: K\ F\ N$
- On aurait alors :

  $(\lambda^{+}x \,.\, I\ I) \equiv K\ (I\ I)$

# Un algorithme d'abstraction $(\lambda^{+++}x.M)$

- Soit x une variable et M un terme, on définit $(\lambda^{+++}x.M)$ par induction sur M :
  - $(\lambda^{+++}x.F) \equiv_{def} K\ F$ si $x \notin F$ (PRIORITAIRE)
  - $(\lambda^{+++}x.F\ x) \equiv_{def} F$ si $x \notin F$ (PRIORITAIRE)
  - $(\lambda^{+++}x.c) \equiv_{def} K\ c$ si c est une constante (BASE)
  - $(\lambda^{+++}x.x) \equiv_{def} I$ (BASE)
  - $(\lambda^{+++}x.y) \equiv_{def} K\ y$ si y est une variable et $y \neq x$ (BASE)
  - $(\lambda^{+++}x.PQ) \equiv_{def} S\ (\lambda^{+++}x.P)\ (\lambda^{+++}x.Q)$

- Théorème. On a :

  $(\lambda^{+++}x.M)\ N := [N / x]\ M$ et $x \notin (\lambda^{+++}x.M)$

# Un algorithme d'abstraction $(\lambda^{++++}x.M)$

- Soit x une variable et M un terme, on définit $(\lambda^{++++}x.M)$ par induction sur M :
  - $(\lambda^{++++}x.\ F\ (G\ x)) \equiv_{def} B\ F\ G$ si $x \notin (F\ G)$ (PRIORITAIRE)
  - $(\lambda^{++++}x.\ F\ x\ (G\ x)) \equiv_{def} S\ F\ G$ si $x \notin (F\ G)$ (PRIORITAIRE)
  - $(\lambda^{++++}x.\ F\ x\ G) \equiv_{def} C\ F\ G$ si $x \notin (F\ G)$ (PRIORITAIRE)
  - $(\lambda^{++++}x.F) \equiv_{def} K\ F$ si $x \notin F$ (PRIORITAIRE)
  - $(\lambda^{++++}x.F\ x) \equiv_{def} F$ si $x \notin F$ (PRIORITAIRE)
  - etc. (W, C*)

- Théorème. On a :

  $(\lambda^{++++}x.M)\ N := [N / x]\ M$ et $x \notin (\lambda^{++++}x.M)$

# Une autre manière d'améliorer ($\lambda^+x.M$)

- La méthode : simplifier le résultat de $(\lambda^+x.M)$ :
  - $S\ (K\ F)\ (K\ G) \rightarrow K\ (F\ G)$ si $x \notin (F\ G)$
  - $S\ (K\ F)\ G \rightarrow B\ F\ G$ si $x \notin (F\ G)$
  - $S\ F\ (K\ G) \rightarrow C\ F\ G$ si $x \notin (F\ G)$
  - etc.

- Ces règles de simplifications ne font pas mieux que les algorithmes améliorés (elles en sont extraites) mais elles sont plus délicates à manier…

# Modéliser les structures de données

- On sait que l'on peut modéliser toutes les structures de données à partir de la paire que nous noterons <a,b>.

- Une *modélisation de la paire* est un triplet de fonctions $\pi$, $\pi_1$, $\pi_2$ tel que :
  - $\pi$ a b est le modèle de la paire ;
  - $\pi_1$ est la première projection, i.e. $\pi_1(\pi\ a\ b) := a$
  - $\pi_2$ est la deuxièmeprojection, i.e. $\pi_2(\pi\ a\ b) := b$

# La paire de CHURCH

- La paire de CHURCH est une modélisation de la paire.
- On pose $\pi \equiv_{def} (\lambda^{+}a.\ (\lambda^{+}b.\ (\lambda^{+}x.\ x\ a\ b)))$
- $\pi\ a\ b$ est égal à une valeur qui ne nous importe pas et $\pi\ a\ b\ x := x\ a\ b$
- Si je veux extraire $a$ de la paire $(\pi\ a\ b)$, il me suffit de l'appliquer à $K$ car $\pi\ a\ b\ K := K\ a\ b := a$. On posera donc $\pi_1 \equiv_{def} (\lambda^{+}c.\ c\ K)$
- Si je veux extraire $b$ de la paire $(\pi\ a\ b)$, il me suffit de l'appliquer à $K\ I$ car $\pi\ a\ b\ (K\ I) := K\ I\ a\ b := I\ b$. On posera donc $\pi_2 \equiv_{def} (\lambda^{+}c.\ c\ (K\ I))$

# Les vecteurs

- Un vecteur de longueur n, $[a_1,..,a_n]$, peut être modélisé à l'aide des paires par

$$<a_1 , <a_2, < \dots , <a_{n-1},<a_n, K\ I>>..>>$$

# Exercice

On cherche à construire conjointement une famille de combinateurs $(B_n)_{n\geqslant 1}$ tels que $B_n f g x_1 \cdots x_n = f(g x_1 \cdots x_n)$ et une famille de combinateurs $(V_n)_{n\geqslant 1}$ tels que :

$$\begin{cases} V_{n+1} x_1 x_2 \cdots x_{n+1} = \pi x_1 \, (V_n x_2 \cdots x_{n+1}) & \text{si} \quad n \geqslant 1 \\ V_1 x_1 = \pi x_1 (K I) \end{cases}$$

1. Déterminer $B_1$.
2. Exprimer $B_{n+1}$ en fonction de $B_n$.
3. Déterminer $V_1$.
4. Exprimer $V_{n+1}$ en fonction de $B_n$ et $V_n$.
5. Exprimer le couple (paire) $\langle V_{n+1}, B_{n+1} \rangle$ en fonction du couple $\langle V_n, B_n \rangle$.
6. En déduire une expression pour $V_n$ ne faisant appel qu'aux combinateurs de base et à l'entier de Church $\underline{n}$.

# Exercice

- Puis cherchez $P_i$ en fonction de $i$ telle que

  $P_i [a_1, ..., a_k] = a_i$ si $1 \leq i \leq k$

# La fonction prédécesseur

- On cherche $\underline{p}$ tel que $\underline{p}\ \underline{0} := \underline{0}$ et $\underline{p}\ \underline{n+1} := \underline{n}$.
- L'opération $\langle x , y\rangle \rightarrow \langle y , y+1\rangle$ est réalisée par le terme $(\lambda^{+}c.\ \pi\ (\pi_2\ c)\ (s\ (\pi_2\ c)))$.
- On itère $k$ fois cette opération sur la paire $\langle 0 , 0\rangle$ :
  - $\langle 0 , 0\rangle$, itération $0$
  - $\langle 0 , 1\rangle$, itération $1$
  - $\langle 1 , 2\rangle$, itération $2$
  - …
  - $\langle k-1 , k\rangle$, itération $k$
- $\underline{p} \equiv_{def} (\lambda^{+}n.\ (\pi_1\ (n\ (\lambda^{+}c.\ \pi\ (\pi_2\ c)\ (s\ (\pi_2\ c)))\ (\pi\ 0\ 0))))$

# En vrac…

- $\underline{-} = (\lambda^{+}x,y\ .\ y\ \underline{p}\ x)$
- $\underline{\leqq} = (\lambda^{+}x,y\ .\ \underline{z}\ (\underline{-}\ x\ y))$
- $\underline{\geqq} = (\lambda^{+}x,y\ .\ \underline{z}\ (\underline{-}\ y\ x))$
- $\underline{>} = (\lambda^{+}x,y\ .\ \underline{\sqsupset}\ (\underline{\leqq}\ x\ y))$
- $\underline{<} = (\lambda^{+}x,y\ .\ \underline{\sqsupset}\ (\underline{\geqq}\ x\ y))$
- $\underline{=} = (\lambda^{+}x,y\ .\ \underline{\triangle}\ (\underline{\leqq}\ x\ y)\ (\underline{\geqq}\ x\ y))$
- $\underline{/} = (\lambda^{+}x,y\ .\ \underline{\leq}\ x\ y\ \underline{0}\ (\underline{s}\ (\underline{/}\ (\underline{-}\ x\ y)\ y)))$
  donc : $\underline{/} = Y\ (\lambda^{+}f\ .\ (\lambda^{+}x,y\ .\ \underline{\leq}\ x\ y\ \underline{0}\ (\underline{s}\ (f\ (\underline{-}\ x\ y)\ y))))$
- $\underline{\%} = (\lambda^{+}x,y\ .\ \underline{\leq}\ x\ y\ \underline{x}\ (\underline{\%}\ (\underline{-}\ x\ y)\ y))$
  donc : $\underline{\%} = Y\ (\lambda^{+}f\ .\ (\lambda^{+}x,y\ .\ \underline{\leq}\ x\ y\ \underline{x}\ (\underline{\%}\ (\underline{-}\ x\ y)\ y)))$
- etc.

# La fonction factorielle

- Définition :

  $f(x) = \mathit{if}\ \ x=0\ \ \mathit{then}\ \ 1\ \ \mathit{else}\ \ x * f(x-1)$

- 1ère étape : curryfication, écriture préfixée

  $f\ x = \underline{z}\ x\ \underline{1}\ (\underline{*}\ x\ (f\ \ (\underline{p}\ x)))$

- 2ème étape : abstraction de l'argument

  $f = (\lambda^{+} x\ .\ \underline{z}\ x\ \underline{1}\ (\underline{*}\ x\ (f\ \ (\underline{p}\ x))))$

- 3ème étape : équation de point-fixe

  $f = (\lambda^{+} f\ .\ (\lambda^{+} x\ .\ \underline{z}\ x\ \underline{1}\ (\underline{*}\ x\ (f\ \ (\underline{p}\ x))))\ f$

- 4ème étape : extraction du point-fixe

  $f = Y\ (\lambda^{+} f\ .\ (\lambda^{+} x\ .\ \underline{z}\ x\ \underline{1}\ (\underline{*}\ x\ (f\ \ (\underline{p}\ x)))))$

# Définitions multiples

- Un programme (fonctionnel) n'est pas simplement fait de définitions explicites ou de définitions implicites simples. En général c'est un ensemble de définitions de fonctions s'appelant les unes les autres.

- Exemple:

  pair(x) = *si* x=0 *alors* true *sinon* impair(x-1)

  impair(x) = *si* x=0 *alors* false *sinon* pair(x-1)

On appelle cela des *récursivités croisées*.

# Définitions multiples

- La forme générale d'un programme est :

$$\begin{cases} f_1\ x_1 \ldots x_n = E_1(f_1,\ldots,f_k,x_1,\ldots,x_n) \\ f_2\ x_1 \ldots x_n = E_2(f_1,\ldots,f_k,x_1,\ldots,x_n) \\ \ldots \\ f_k\ x_1 \ldots x_n = E_k(f_1,\ldots,f_k,x_1,\ldots,x_n) \end{cases}$$

- *Il n'est pas nécessaire que les fonctions aient la même arité mais j'en ai marre de cliquer dans cet éditeur PPP pour obtenir des indices d'indices... Bref...*

# Définitions multiples

- On commence par réaliser l'abstraction des variables dans les corps des fonctions :

$$\begin{cases} f_1 = \lambda^+ x_1 \ldots x_n \,.\, E_1(f_1,\ldots,f_k,x_1,\ldots,x_n) \\ f_2 = \lambda^+ x_1 \ldots x_n \,.\, E_2(f_1,\ldots,f_k,x_1,\ldots,x_n) \\ \ldots \\ f_k = \lambda^+ x_1 \ldots x_n \,.\, E_k(f_1,\ldots,f_k,x_1,\ldots,x_n) \end{cases}$$

# Définitions multiples

- Puis on écrit :

$$\begin{cases} f_1 = (\lambda^+ f_1 \dots f_k \,.\, (\lambda^+ x_1 \dots x_n \,.\, E_1(f_1,\dots,f_k,x_1,\dots,x_n))\ f_1 \dots f_k \\ f_2 = (\lambda^+ f_1 \dots f_k \,.\, (\lambda^+ x_1 \dots x_n \,.\, E_2(f_1,\dots,f_k,x_1,\dots,x_n))\ f_1 \dots f_k \\ \dots \\ f_k = (\lambda^+ f_1 \dots f_k \,.\, (\lambda^+ x_1 \dots x_n \,.\, E_k(f_1,\dots,f_k,x_1,\dots,x_n))\ f_1 \dots f_k \end{cases}$$

Soit :

$$\begin{cases} f_1 = U_1\ f_1 \dots f_k,\ U_1 \equiv_{def} (\lambda^+ f_1 \dots f_k \,.\, (\lambda^+ x_1 \dots x_n \,.\, E_1(f_1,\dots,f_k,x_1,\dots,x_n)) \\ f_2 = U_2\ f_1 \dots f_k,\ U_2 \equiv_{def} (\lambda^+ f_1 \dots f_k \,.\, (\lambda^+ x_1 \dots x_n \,.\, E_2(f_1,\dots,f_k,x_1,\dots,x_n)) \\ \dots \\ f_k = U_k\ f_1 \dots f_k,\ U_k \equiv_{def} (\lambda^+ f_1 \dots f_k \,.\, (\lambda^+ x_1 \dots x_n \,.\, E_k(f_1,\dots,f_k,x_1,\dots,x_n)) \end{cases}$$

# Définitions multiples

Puis on s'intéresse à :

$$F = [f_1, f_2, \ldots, f_k]$$

$$= V_k\ f_1\ f_2 \ldots f_k$$

$$= V_k\ (U_1\ f_1\ f_2 \ldots f_k)\ (U_2\ f_1\ f_2 \ldots f_k) \ldots (U_k\ f_1\ f_2 \ldots f_k)$$

$$= V_k\ (U_1\ (P_1\ F)\ (P_2\ F) \ldots (P_k\ F)) \ldots (U_k\ (P_1\ F)\ (P_2\ F) \ldots (P_k\ F))$$

Là, on dispose d'une équation simple en F que l'on résoud comme déjà vu :

$$F = Y\ (\lambda^+\ F.\ V_k\ (U_1\ (P_1\ F)\ (P_2\ F) \ldots (P_k\ F)) \ldots (U_k\ (P_1\ F)\ (P_2\ F) \ldots (P_k\ F)))$$

Puis :

$$f_i \equiv_{def} P_i\ F \quad \text{pour } i \text{ de } 1 \text{ à } k$$

# Base de combinateurs

- Un ensemble {C1,..,Cn} de combinateurs est appelé une *base* si pour toute variable x et tout terme M il existe un terme F construit avec les combinateurs de la base uniquement et tel que :

$$F\ N = [N / x]\ M$$

- Exemple : {S,K} est une base (algorithme $\lambda^+$).
- Exemple : {B,C,K,W} est une base.
  *A démontrer en exercice.*

# Un autre pour le bestiaire…

- On pose $Q = (\lambda^{+}x \,.\, x\ K\ S\ K)$
- $Q\ Q = Q\ K\ S\ K = K\ K\ S\ K\ S\ K = K\ K\ S\ K = K\ K$
- $Q\ Q\ Q = K\ K\ Q = K$
- $Q\ (Q\ Q) = Q\ (K\ K) = K\ K\ K\ S\ K = K\ S\ K = S$

- On peut exprimer S et K en fonction de Q donc :
  { Q } est une base !

- Un seul combinateur suffit-il ? Non ! Où est la faille ?

# Combinateur propre

- Un combinateur C est dit propre s'il existe n tel que C $x_1,..,x_n$ := E où E est une combinaison pure de $x_1,..,x_n$, c'est-à-dire un terme construit à partir de $x_1,..,x_n$ uniquement par l'opération d'application.
- Exemple : S est propre (n=3).
- Exemple : K est propre (n=2).
- Exemple : Q est impropre.
- Un combinateur impropre ne peut être défini par une règle de réduction. Il a « besoin » d'autres combinateurs propes et pré-définis.

# Théorème de CRAIG (1958, grande année !)

- Une base de combinateurs propres doit contenir au moins deux éléments.
- La démonstration est basée sur les *effets* des combinateurs (quelques pages) :
  - effet d'*effacement* : K
  - effet *associatif* : S, B
  - effet *permutatif* : S, C
  - effet *duplicatif* : S, W
- Une base de combinateurs doit proposer ces 4 effets.
- Un seul combinateur ne peut avoir ces 4 effets.

# Combinateurs et calculabilité

Il est clair que les fonctions partielles récursives peuvent être implémentées par S et K donc (thèse de CHURCH) :

Tout ce qui est
effectivement calculable
peut être calculé par des combinateurs.

# Combinateurs et implémentation

- Le travail de cette section montre que l'on peut implémenter un langage de programmation avec S et K.
- Une fois que la possibilité de modéliser quelque chose est démontrée, on peut l'incorporer sans risque dans la théorie. Exemple : on introduit des constantes pour les booléens et leurs opérations et des règles appropriées. Ces règles qui ont une portée sémantique sont appelées des δ–règles.
- Langage implémenté par des combinateurs :
  - HASKELL, MIRANDA, etc.
  - CAML mais avec des combinateurs bien plus puissants.

# Combinateurs et programmation

- Penser en termes combinateurs signifie penser en termes de combinaisons d'opérateurs, d'opérateurs sur les opérateurs, etc.
- Des langages : FP, GRAAL, …
- Un style de programmation : composer des opérations…
- Une maîtrise des processus récursifs : programmer la fonction de Fibonacci avec un langage de macro-expansion comme M4.
- Une culture… Formalisme. Opérateurs. Abstraction.