Lambda-calcul et sémantique des langages de programmation

Thibaut Balabonski @ *UPSay*
`https://www.lri.fr/~blsk/Semantique/`
Hiver 2021

Chapitre 1 - Le λ-calcul

1 Théorie des fonctions

Repères

1870 Sur quels objets bâtir les mathématiques ? Ensembles ou fonctions ?

1920 Moses Schönfinkel, Haskell Curry : logique combinatoire Briques de base pour fabriquer des fonctions

1936 Alonzo Church : λ-calcul Caractérisation de ce qui est calculable par algorithme Équivalent aux machines de Turing Résolution de l'*Entscheidungsproblem*

1970+ Essor du λ-calcul avec l'informatique Programmation fonctionnelle Assistants à la preuve

Fonctions
Notations variées pour un même concept

Maths	$x \mapsto x$
	$f \mapsto (x \mapsto f(f(x)))$
Caml	`fun x -> x`
	`fun f -> (fun x -> f(f x))`
Python	`lambda x: x`
	`lambda f: (lambda x: f(f(x)))`
λ-calcul	$\lambda x.x$
	$\lambda f.(\lambda x.f(f\ x))$

2 Aperçu du λ-calcul

Le λ-calcul est défini par un ensemble de *termes*, représentant des programmes ou algorithmes, et par des *règles de conversion*, décrivant l'évolution des calculs.

Termes (expressions)
La syntaxe du λ-calcul est incarnée par une notion d'*expression*, ou *terme*. Les termes sont formés avec trois constructions.

x variable, référence au paramètre d'une fonction

$t_1 t_2$ application d'un terme t_1 à un terme t_2, t_1 peut être compris comme une fonction et t_2 comme son argument

$\lambda x.t$ fonction d'un paramètre x, dont le résultat est donné par t

Approche *intentionnelle* : on décrit les fonctions par leur comportement

Exemples de termes

— Fonction identité

$$\lambda x.x$$

prend un paramètre x et renvoie la valeur de x

— Générateur de fonctions constantes

$$\lambda c.(\lambda x.c)$$

prend un paramètre c et renvoie la fonction constante dont le résultat est systématiquement c

— Distribution

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

prend un paramètre x et on verra le reste plus tard

— Paradoxe ?

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

prend un paramètre x et l'applique à lui-même

Notations

— Au lieu de $\lambda x_1.(\ldots(\lambda x_n.t)\ldots)$ on écrit

$$\lambda x_1 \ldots x_n.t$$

— Au lieu de $(\ldots(t\ u_1)\ldots u_n)$ on écrit

$$t\ u_1 \ldots u_n$$

ou $t\ \vec{u}$ avec $\vec{u} = u_1 \ldots u_n$

D'où

$\lambda c.(\lambda x.c)$	$\lambda cx.c$
$\lambda x.(\lambda y.(\lambda z.((x\ z)\ (y\ z))))$	$\lambda xyz.xz(yz)$

Fonctions n-aires et curryfication

En λ-calcul, pas de notion de produit cartésien.

— Pour représenter $(x, y) \mapsto t$ on écrit

$$\lambda x.\lambda y.t \qquad \text{ou} \qquad \lambda xy.t$$

— Pour représenter $f(x, y)$ on écrit

$$f\ x\ y$$

Les fonctions sont dites *curryfiées* (de Haskell Curry)
Cette présentation permet les *applications partielles*

Calculer avec le λ-calcul
Unité de calcul : fonction appliquée à un argument

$$(\lambda x.t)\ u \quad \rightarrow \quad t\{x \leftarrow u\}$$

Résultat :

t où chaque occurrence de x est remplacée par u $\mathrm{t}\{x \leftarrow u\}$

Exemple de calcul

$$\begin{aligned}
&(\lambda xyz.xz\ (yz))\ (\lambda ab.a)\ t\ u && \\
&\rightarrow\ (\lambda yz.(\lambda ab.a)z\ (yz))\ t\ u && \{x \leftarrow \lambda ab.a\} \\
&\rightarrow\ (\lambda z.(\lambda ab.a)z\ (tz))\ u && \{y \leftarrow t\} \\
&\rightarrow\ (\lambda ab.a)u\ (tu) && \{z \leftarrow u\} \\
&\rightarrow\ (\lambda b.u)\ (tu) && \{a \leftarrow u\} \\
&\rightarrow\ u && \{b \leftarrow tu\}
\end{aligned}$$

Exercice : réductions
Calculer le résultat du terme

$$(\lambda xy.yx)\ (\lambda ab.b)\ (\lambda s.stu)$$

Correction

$$\begin{aligned}
&(\lambda xy.yx)\ (\lambda ab.b)\ (\lambda s.stu) \\
&\rightarrow\ (\lambda y.y\ (\lambda ab.b))\ (\lambda s.stu) \\
&\rightarrow\ (\lambda s.stu)\ (\lambda ab.b) \\
&\rightarrow\ (\lambda ab.b)\ t\ u \\
&\rightarrow\ (\lambda b.b)\ u \\
&\rightarrow\ u
\end{aligned}$$

Exercice : logique combinatoire
La logique combinatoire (Schönfinkel, 1920 - Curry, 1930) est basée sur des symboles I, K, S, B, C (appelés combinateurs) associés aux règles suivantes

$$\begin{aligned}
I\ x &\rightarrow x \\
K\ x\ y &\rightarrow x \\
S\ x\ y\ z &\rightarrow xz\ (yz) \\
B\ x\ y\ z &\rightarrow x\ (yz) \\
C\ x\ y\ z &\rightarrow xz\ y
\end{aligned}$$

Donner des λ-termes équivalents à ces combinateurs
Calculer les résultats des expressions suivantes

1. $S\ K\ K\ x$
2. $S\ (K\ S)\ K$

Correction λ-termes équivalents aux combinateurs

— $I = \lambda x.x$

— $K = \lambda xy.x$

— $S = \lambda xyz.xz(yz)$

— $B = \lambda xyz.x(yz)$

— $C = \lambda xyz.xzy$

Réductions

— $S\ K\ K$ est équivalent à I

$$\begin{array}{rcl} S\ K\ K\ x & \rightarrow & Kx(Kx) \\ & \rightarrow & x \end{array}$$

— $S\ (K\ S)\ K$ est équivalent à B

$$\begin{array}{rcl} S\ (K\ S)\ K\ x\ y\ z & \rightarrow & (K\ S\ x)\ (K\ x)\ y\ z \\ & \rightarrow & S\ (K\ x)\ y\ z \\ & \rightarrow & (K\ x\ z)\ (y\ z) \\ & \rightarrow & x\ (y\ z) \end{array}$$

Remplacements suspects / phénomème de capture
Comment résoudre les remplacements suivants ?

$$(\lambda x.(\lambda x.x))\ y \quad \rightarrow \quad (\lambda x.x)\{x \leftarrow y\}$$

$$(\lambda x.(\lambda y.x))\ y \quad \rightarrow \quad (\lambda y.x)\{x \leftarrow y\}$$

Autrement dit : quelle est la portée des variables ?

3 Formalisation des λ-termes

Ensemble des termes
L'ensemble Λ des λ-termes est *le plus petit ensemble* qui contient :

1. x pour toute variable x
2. $\lambda x.t$ si $t \in \Lambda$
3. $t_1\ t_2$ si $t_1 \in \Lambda$ et $t_2 \in \Lambda$

Notation sous la forme d'une grammaire

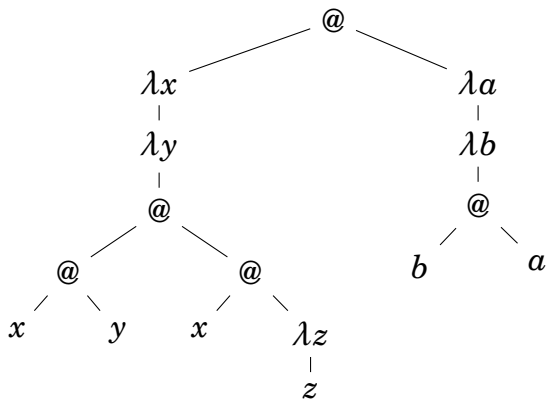
$$t \ ::= \ x \ | \ \lambda x.t \ | \ t_1\ t_2$$

C'est une définition par clôture : elle *permet la récurrence*

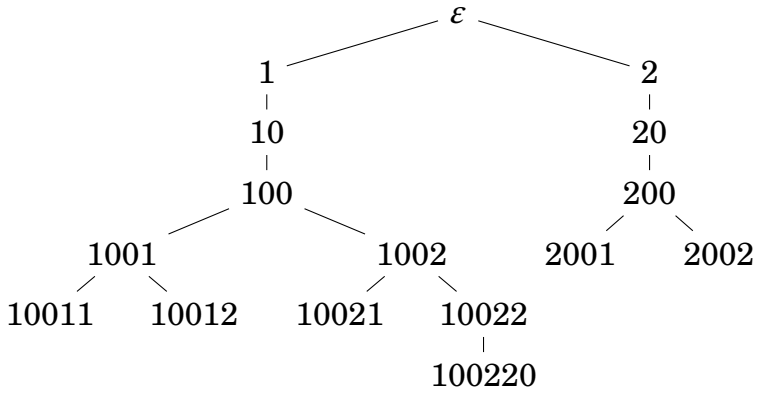
Un terme est un arbre
L'écriture

$$(\lambda xy.xy(x(\lambda z.z))\ (\lambda ab.ba)$$

désigne l'arbre



Positions d'un terme
Position : mot sur l'alphabet $\{0, 1, 2\}$ désignant un chemin depuis la racine



Ensemble $\mathsf{pos}(t)$ des positions du terme t

$$\begin{array}{rcl} \mathsf{pos}(x) & = & \{\varepsilon\} \\ \mathsf{pos}(\lambda x.t) & = & \{\varepsilon\} \quad \cup \quad 0 \cdot \mathsf{pos}(t) \\ \mathsf{pos}(t_1\ t_2) & = & 1 \cdot \mathsf{pos}(t_1) \quad \cup \quad 2 \cdot \mathsf{pos}(t_2) \end{array}$$

Représentation en caml
Un type pour représenter les λ-termes

```
type terme =
  | Var of string
  | Abs of string * terme
  | App of terme  * terme
```

Représentation du terme $\lambda ab.ba$

```
Abs("a", Abs("b", App(Var "b", Var "a")))
```

Définition d'une fonction sur Λ

Définition de f par récurrence, en trois cas

— $f(x)$ cas de base

— $f(\lambda x.t)$ en utilisant $f(t)$

— $f(t_1\ t_2)$ en utilisant $f(t_1)$ et $f(t_2)$

Exemples

$f_@$: nombre d'applications

$$\begin{array}{rcl} f_@(x) & = & 0 \\ f_@(\lambda x.t) & = & f_@(t) \\ f_@(t_1\ t_2) & = & 1 + f_@(t_1) + f_@(t_2) \end{array}$$

f_v : nombre de variables

$$\begin{array}{rcl} f_v(x) & = & 1 \\ f_v(\lambda x.t) & = & f_v(t) \\ f_v(t_1\ t_2) & = & f_v(t_1) + f_v(t_2) \end{array}$$

Définition d'une fonction en caml

Programmation de $f_@$

```
let rec nb_app = function
  | Var _        -> 0
  | Abs(_, t)    -> nb_app t
  | App(t1, t2) -> 1 + nb_app t1 + nb_app t2
```

Programmation de f_v

```
let rec nb_var = function
  | Var _        -> 1
  | Abs(_, t)    -> nb_var t
  | App(t1, t2) -> nb_var t1 + nb_var t2
```

Principe de récurrence sur Λ

Justification qu'une proposition P est vraie pour tous les λ-termes, en trois cas

— justifier $P(x)$ pour toute variable x

— justifier $P(\lambda x.t)$ en supposant $P(t)$ vraie

— justifier $P(t_1\ t_2)$ en supposant $P(t_1)$ et $P(t_2)$ vraies

Exemple de raisonnement par récurrence

À prouver : pour tout $t \in \Lambda$, $f_v(t) = 1 + f_@(t)$

— Justification de $P(x)$. Par définition $f_v(x) = 1$ et $f_@(x) = 0$ Donc $f_v(x) = 1 + f_@(x)$

— Justification de $P(t) \Rightarrow P(\lambda x.t)$. Supposons $f_v(t) = 1 + f_@(t)$. Alors

$$\begin{array}{rcll} f_v(\lambda x.t) & = & f_v(t) & \textit{par définition de } f_v \\ & = & 1 + f_@(t) & \textit{par hypothèse de récurrence} \\ & = & 1 + f_@(\lambda x.t) & \textit{par définition de } f_@ \end{array}$$

— Justification de $P(t_1) \wedge P(t_2) \Rightarrow P(t_1\ t_2)$. Supposons $f_v(t_1) = 1 + f_@(t_1)$ et $f_v(t_2) = 1 + f_@(t_2)$. Alors

$$\begin{array}{rcll} & & f_v(t_1\ t_2) & \\ & = & f_v(t_1) + f_v(t_2) & \textit{par définition de } f_v \\ & = & 1 + f_@(t_1) + 1 + f_@(t_2) & \textit{par hypothèses de récurrence} \\ & = & 1 + (1 + f_@(t_1) + f_@(t_2)) & \\ & = & 1 + f_@(t_1\ t_2) & \textit{par définition de } f_@ \end{array}$$

4 Variables et substitutions

À propos des variables
L'abstraction

$$\lambda x.t$$

introduit une variable x *locale* à t On parle aussi de *variable liée*
Autrement dit

— le nom de x est inconnu de l'extérieur
— vu de l'extérieur, le nom x importe peu
— changer le nom x est transparent pour l'extérieur

Variables libres
Variables qui sont visibles de l'extérieur

$$\begin{array}{rcl} \mathsf{fv}(x) & = & \{x\} \\ \mathsf{fv}(t_1\ t_2) & = & \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2) \\ \mathsf{fv}(\lambda x.t) & = & \mathsf{fv}(t) \setminus \{x\} \end{array}$$

Terme sans variable libre : *terme clos*, ou *combinateur*
Variable à la fois libre et liée

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

Substitution
Remplacement dans t des occurrences libres de x par u.

$$t\{x \leftarrow u\}$$

Définition par récurrence sur la structure de t.

$$\begin{array}{rcl} y\{x \leftarrow u\} & = & \begin{cases} u & \text{si } x = y \\ y & \text{si } x \neq y \end{cases} \\ (t_1\ t_2)\{x \leftarrow u\} & = & t_1\{x \leftarrow u\}\ t_2\{x \leftarrow u\} \\ (\lambda y.t)\{x \leftarrow u\} & = & \begin{cases} \lambda y.t & \text{si } x = y \\ \lambda y.t\{x \leftarrow u\} & \text{si } x \neq y \text{ et } y \notin \mathsf{fv}(u) \\ \lambda z.t\{y \leftarrow z\}\{x \leftarrow u\} & \text{si } x \neq y \text{ et } y \in \mathsf{fv}(u) \\ & z \text{ nouvelle variable} \end{cases} \end{array}$$

Convention de Barendregt
Pour simplifier, on ne considère que des termes dans lesquels

aucune variable n'apparaît à la fois libre et liée, dans aucun sous-terme

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

Substitution simplifiée grâce à la convention

$$\begin{array}{rcl} y\{x \leftarrow u\} & = & \begin{cases} u & \text{si } x = y \\ y & \text{si } x \neq y \end{cases} \\ (t_1\ t_2)\{x \leftarrow u\} & = & t_1\{x \leftarrow u\}\ t_2\{x \leftarrow u\} \\ (\lambda y.t)\{x \leftarrow u\} & = & \lambda y.t\{x \leftarrow u\} \end{array}$$

(In)stabilité de la convention de Barendregt

$$\begin{array}{l}(\lambda x.xx)\ (\lambda yz.yz)\\ \rightarrow\ \ (\lambda yz.yz)\ (\lambda yz.yz)\\ \rightarrow\ \ \lambda z.((\lambda yz.zy)z)\end{array}$$

Pour préserver la convention de Barendregt : *changer le nom de certaines variables en cours de calcul*

Renommage des variables liées : α-conversion

$$\lambda x.t \quad =_\alpha \quad \lambda y.(t\{x \leftarrow y\}) \qquad\qquad \text{si } y \notin \mathsf{fv}(t)$$

L'α-conversion ne change pas la signification d'un terme

— on peut l'appliquer *quand* on veut

L'α-conversion est une *congruence*

$$\begin{array}{rcl} t =_\alpha t' & \Longrightarrow & \lambda x.t =_\alpha \lambda x.t' \\ t_1 =_\alpha t'_1 & \Longrightarrow & t_1\ t_2 =_\alpha t'_1\ t_2 \\ t_2 =_\alpha t'_2 & \Longrightarrow & t_1\ t_2 =_\alpha t_1\ t'_2 \end{array}$$

— on peut l'appliquer *où* on veut

On supposera dorénavant que tous les termes vérifient la convention de Barendregt

Exercice : variables liées et renommage
Renommer les variables des termes suivants afin qu'ils respectent la convention de Barendregt

1. $\lambda x.(\lambda x.xy)(\lambda y.xy)$
2. $\lambda xy.x(\lambda y.(\lambda y.y)yz)$

Calculer le résultat du terme

$$(\lambda f.f\ f)\ (\lambda ab.b\ a\ b)$$

Correction

1. $\lambda x.(\lambda x.xy)(\lambda y.xy) \quad =_\alpha \quad \lambda x.(\lambda z.zy)(\lambda t.xt)$
2. $\lambda xy.x(\lambda y.(\lambda y.y)yz) \quad =_\alpha \quad \lambda xy.x(\lambda a.(\lambda b.b)az)$
3. $$\begin{array}{rcl}(\lambda f.f\ f)\ (\lambda ab.b\ a\ b) & \rightarrow_\beta & (\lambda ab.b\ a\ b)\ (\lambda ab.b\ a\ b)\\ & \rightarrow_\beta & \lambda ab.b\ (\lambda ab.b\ a\ b)\ b\\ & =_\alpha & \lambda b.b\ (\lambda xy.y\ x\ y)\ b\end{array}$$

Exercice : variables libres et substitution
Démontrer que

$$\mathsf{fv}(t\{x \leftarrow u\}) \quad \subseteq \quad (\mathsf{fv}(t) \setminus \{x\}) \cup \mathsf{fv}(u)$$

A-t-on égalité entre ces deux ensembles ?
Correction Démonstration par récurrence sur la structure de t

— Cas où t est une variable
  — cas x : $\mathsf{fv}(x\{x \leftarrow u\}) = \mathsf{fv}(u) \subseteq (\mathsf{fv}(t) \setminus \{x\}) \cup \mathsf{fv}(u)$
  — cas $y \neq x$: $\mathsf{fv}(y\{x \leftarrow u\}) = \mathsf{fv}(y) = \{y\}$, et $\{y\}$ est bien inclus dans $(\mathsf{fv}(y) \setminus \{x\}) \cup \mathsf{fv}(u) = \{y\} \cup \mathsf{fv}(u)$

— Cas où t est une application $t_1\ t_2$. On suppose $\mathsf{fv}(t_1\{x \leftarrow u\}) \quad \subseteq \quad (\mathsf{fv}(t_1) \setminus \{x\}) \cup \mathsf{fv}(u)$ et $\mathsf{fv}(t_2\{x \leftarrow u\}) \quad \subseteq \quad (\mathsf{fv}(t_2) \setminus \{x\}) \cup \mathsf{fv}(u)$ (ce sont les hypothèses de récurrence). Alors

$$\begin{array}{lll}
\mathsf{fv}((t_1\ t_2)\{x \leftarrow u\}) & & \\
= \quad \mathsf{fv}((t_1\{x \leftarrow u\})\ (t_2\{x \leftarrow u\})) & & \text{par définition de la substitution} \\
= \quad \mathsf{fv}(t_1\{x \leftarrow u\}) \cup \mathsf{fv}(t_2\{x \leftarrow u\}) & & \text{par définition de fv} \\
\subseteq \quad (\mathsf{fv}(t_1) \setminus \{x\}) \cup \mathsf{fv}(u) \cup (\mathsf{fv}(t_2) \setminus \{x\}) \cup \mathsf{fv}(u) & & \text{par hypothèses de récurrence} \\
= \quad (\mathsf{fv}(t_1) \setminus \{x\}) \cup (\mathsf{fv}(t_2) \setminus \{x\}) \cup \mathsf{fv}(u) & & \\
= \quad ((\mathsf{fv}(t_1) \cup \mathsf{fv}(t_2)) \setminus \{x\}) \cup \mathsf{fv}(u) & & \\
= \quad (\mathsf{fv}(t_1\ t_2) \setminus \{x\}) \cup \mathsf{fv}(u) & &
\end{array}$$

— Cas où t est une abstraction $\lambda y.t_0$. On suppose $x \neq y$ et $y \notin \mathsf{fv}(u)$ (sinon α-renommage). On suppose $\mathsf{fv}(t_0\{x \leftarrow u\}) \quad \subseteq \quad (\mathsf{fv}(t_0) \setminus \{x\}) \cup \mathsf{fv}(u)$ (hypothèse de récurrence). Alors

$$\begin{array}{lll}
\mathsf{fv}((\lambda y.t_0)\{x \leftarrow u\}) & & \\
= \quad \mathsf{fv}(\lambda y.(t_0\{x \leftarrow u\})) & & \text{car } x \neq y \text{ et } y \notin \mathsf{fv}(u) \\
= \quad \mathsf{fv}(t_0\{x \leftarrow u\}) \setminus \{y\} & & \\
\subseteq \quad ((\mathsf{fv}(t_0) \setminus \{x\}) \cup \mathsf{fv}(u)) \setminus y & & \text{par hypothèse de récurrence} \\
= \quad ((\mathsf{fv}(t_0) \setminus \{x\} \setminus \{y\}) \cup (\mathsf{fv}(u) \setminus y) & & \\
= \quad ((\mathsf{fv}(t_0) \setminus \{x\} \setminus \{y\}) \cup \mathsf{fv}(u) & & \text{car } y \notin \mathsf{fv}(u) \\
= \quad ((\mathsf{fv}(t_0) \setminus \{y\} \setminus \{x\}) \cup \mathsf{fv}(u) & & \\
= \quad (\mathsf{fv}(\lambda y.t_0) \setminus x) \cup \mathsf{fv}(u) & &
\end{array}$$

On n'a pas égalité : si $x \notin \mathsf{fv}(t)$ alors u disparaît dans $t\{x \leftarrow u\}$ et ses variables libres avec.

5 Formalisation de la réduction

β-réduction
Application d'une fonction à un argument

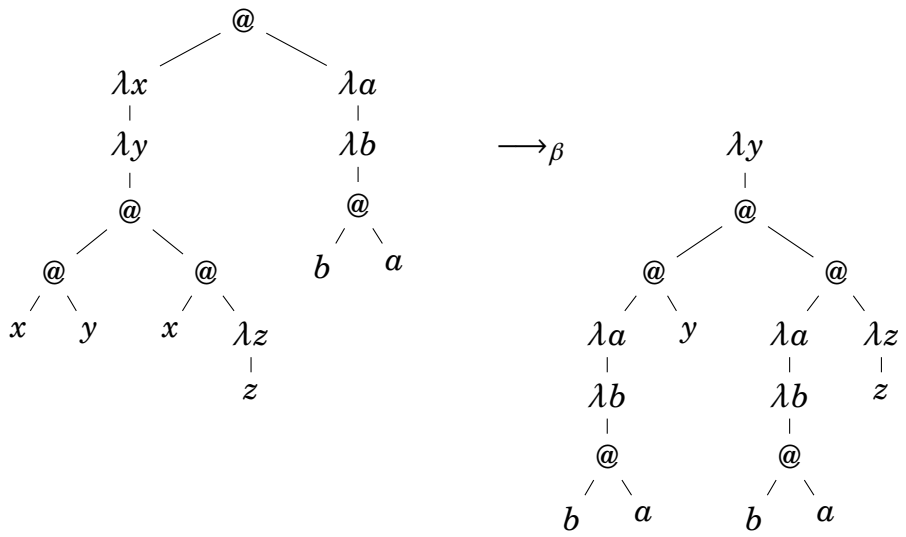
$$(\lambda x.t)\ u$$

Le résultat est donné par le corps de la fonction, dans lequel on lie le paramètre formel x à l'argument u.

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

la notation $t\{x \leftarrow u\}$ désignant la substitution *sans capture*

β-réduction, vue sur les arbres



β-réduction, vue en caml

Fonction réduisant un radical β

```
let beta_reduction = function
  | App(Abs(x, t), u) -> subst t x u
  | _ -> failwith "pas un radical"
```

Fonction auxiliaire : `subst t x u` calcule $t\{x \leftarrow u\}$

```
let rec subst t x u = match t with
  | Var y       -> if x = y then u else t
  | App(t1, t2) -> App(subst t1 x u,
                       subst t2 x u)
  | Abs(y, t)   -> (* renommage ? *)
```

Congruence

La règle de β-réduction peut être appliquée n'importe où dans un terme. Explicitation de ce principe par des règles d'inférence.

$$\frac{}{(\lambda x.t)\ u \quad \to_\beta \quad t\{x \leftarrow u\}}$$

$$\frac{t \quad \to_\beta \quad t'}{t\ u \quad \to_\beta \quad t'\ u} \qquad \frac{u \quad \to_\beta \quad u'}{t\ u \quad \to_\beta \quad t\ u'}$$

$$\frac{t \quad \to_\beta \quad t'}{\lambda x.t \quad \to_\beta \quad \lambda x.t'}$$

Localisation des réductions
On note

$$t \quad \xrightarrow{p}_\beta \quad t'$$

si t se réduit en t' par contraction d'un radical à la position p

$$\frac{}{(\lambda x.t)\ u \quad \xrightarrow{\varepsilon}_\beta \quad t\{x \leftarrow u\}}$$

$$\frac{t \quad \xrightarrow{p}_\beta \quad t'}{t\ u \quad \xrightarrow{1\cdot p}_\beta \quad t'\ u} \qquad \frac{u \quad \xrightarrow{p}_\beta \quad u'}{t\ u \quad \xrightarrow{2\cdot p}_\beta \quad t\ u'}$$

$$\frac{t \quad \xrightarrow{p}_\beta \quad t'}{\lambda x.t \quad \xrightarrow{0\cdot p}_\beta \quad \lambda x.t'}$$

Justification d'une réduction par un arbre de dérivation

$$\dfrac{\dfrac{\dfrac{}{(\lambda y.zy)\ x \quad \xrightarrow{\varepsilon} \quad zx}}{x\ ((\lambda y.zy)\ x) \quad \xrightarrow{2} \quad x\ (zx)}}{\lambda x.(x\ ((\lambda y.zy)\ x)) \quad \xrightarrow{02} \quad \lambda x.(x\ (zx))}}{(\lambda x.x\ ((\lambda y.zy)x))\ z \quad \xrightarrow{102} \quad (\lambda x.x\ (zx))\ z}$$

Raisonnement par récurrence sur la réduction
La relation de réduction $t \to_\beta t'$ étant définie par des règles d'inférence (et donc par clôture) elle est associée à un principe de récurrence. Pour démontrer qu'une propriété P est telle que

$$\forall t, t', \quad t \to_\beta t' \quad \Longrightarrow \quad P(t, t')$$

il suffit de vérifier les quatre points suivants

— $P((\lambda x.t)u,\ t\{x \leftarrow u\})$ pour tous x, t et u *cas de base*
— $P(tu,\ t'u)$ pour tous t, t' et u tels que $P(t, t')$ *cas héréditaire 1*
— $P(tu,\ tu')$ pour tous t, u et u' tels que $P(u, u')$ *cas héréditaire 2*
— $P(\lambda x.t,\ \lambda x.t')$ pour tous x, t et t' tels que $P(t, t')$ *cas héréditaire 3*

Notez le parallèle entre ces quatre conditions et les quatre règles d'inférence

Raisonnement par récurrence sur la réduction
La réduction ne crée pas de variables libres.

$$\text{Si} \quad t \to t' \quad \text{alors} \quad \mathsf{fv}(t') \subseteq \mathsf{fv}(t)$$

Preuve par récurrence sur la dérivation de $t \to t'$.

— Cas $(\lambda x.t)\ u \to t\{x \leftarrow u\}$. On a déjà montré : $\mathsf{fv}(t\{x \leftarrow u\}) \subseteq (\mathsf{fv}(t) \setminus \{x\}) \cup \mathsf{fv}(u)$ Or

$$\begin{array}{rcl} \mathsf{fv}((\lambda x.t)\ u) & = & \mathsf{fv}(\lambda x.t) \cup \mathsf{fv}(u) \\ & = & (\mathsf{fv}(t) \setminus \{x\}) \cup \mathsf{fv}(u) \end{array}$$

— Cas $t\ u \to t'\ u$ avec $t \to t'$. Alors

$$\begin{array}{rcll} \mathsf{fv}(t'\ u) & = & \mathsf{fv}(t') \cup \mathsf{fv}(u) & \text{par définition} \\ & \subseteq & \mathsf{fv}(t) \cup \mathsf{fv}(u) & \text{par hypothèse de récurrence} \\ & = & \mathsf{fv}(t\ u) & \text{par définition} \end{array}$$

— Cas $t\ u' \to t\ u'$ avec $u \to u'$ identique.

— Cas $\lambda x.t \to \lambda x.t'$ avec $t \to t'$. Alors

$$\begin{array}{rcll} \mathsf{fv}(\lambda x.t') & = & \mathsf{fv}(t') \setminus \{x\} & \text{par définition} \\ & \subseteq & \mathsf{fv}(t) \setminus \{x\} & \text{par hypothèse de récurrence} \\ & = & \mathsf{fv}(\lambda x.t) & \text{par définition} \end{array}$$

Séquences de réduction

$\to_\beta$ un pas de réduction

$\to_\beta^*$ clôture réflexive et transitive : 0, 1 ou plusieurs pas

$\leftrightarrow_\beta$ clôture symétrique : un pas dans un sens ou dans l'autre

$=_\beta$ clôture réflexive, symétrique et transitive (équivalence)

Autre règle parfois ajoutée : η
Selon les objectifs, utilisable dans les deux sens

— η-contraction

$$\lambda x.(t\ x) \quad \to_\eta \quad t$$

— η-expansion

$$t \quad \to_\eta \quad \lambda x.(t\ x)$$

Lié à la notion d'*égalité extensionnelle* (égalité de Leibniz)

Représentation alternative : contextes
Isoler le radical r réduit dans un terme t

$$t = C[r] \qquad \to \qquad C[r'] = t'$$

avec $r = (\lambda x.u)v$ et $r' = u\{x \leftarrow v\}$
C est un *contexte* : un terme muni d'un trou

$$C \ ::= \ \Box \ \mid \ C\ t \ \mid \ t\ C \ \mid \ \lambda x.C$$

$C[u]$ est le *remplissage* du trou de C par le terme u

Exercice : décompositions contexte/sous-terme
Voici quelques décompositions de $\lambda x.(x\ \lambda y.xy)$ sous la forme $C[u]$

C	$\square$	$\lambda x.\square$	$\lambda x.(\square\ \lambda y.xy)$	$\lambda x.(x\ \square)$	...
u	$\lambda x.(x\ \lambda y.xy)$	$x\ \lambda y.xy$	x	$\lambda y.xy$	...

Quelles sont les autres décompositions possibles ?
On a déjà vu que

$$(\lambda x.x\ ((\lambda y.zy)x))\ z \quad \to \quad (\lambda x.x\ (zx))\ z$$

Quels sont le contexte et le radical associés à cette réduction ?
Correction Autres décompositions de $\lambda x.(x\ \lambda y.xy)$

C	$\lambda x.(x\ (\lambda y.\square))$	$\lambda x.(x\ (\lambda y.\square\ y))$	$\lambda x.(x\ (\lambda y.x\ \square))$
u	xy	x	y

Décomposition de la réduction :

$$C[(\lambda y.zy)x] \quad \to \quad C[zx]$$

avec $C = (\lambda x.x\ \square)\ z$

Exercice : équivalence des présentations (sens direct)
Montrer que si

$$t \to_\beta t'$$

alors il existe C, x, u, v tels que

$$t = C[(\lambda x.u)v] \qquad \text{et} \qquad t' = C[u\{x \leftarrow v\}]$$

Correction Démonstration par récurrence sur la dérivation de $t \to_\beta t'$.

- Cas de base $t = (\lambda x.u)v \to_\beta u\{x \leftarrow v\} = t'$. Conclusion immédiate avec le contexte $\square$
- Cas $t = t_1 t_2 \to_\beta t'_1 t_2 = t'$ avec $t_1 \to_\beta t'_1$. On suppose qu'il existe C_1, x, u et v tels que $t_1 = C_1[(\lambda x.u)v]$ et $t'_1 = C_1[u\{x \leftarrow v\}]$ (hypothèse de récurrence). Alors il suffit de prendre $C = C_1\ t_2$
- Cas $t = t_1 t_2 \to_\beta t_1 t'_2 = t'$ avec $t_2 \to_\beta t'_2$ similaire, avec contexte $C = t_1\ C_2$
- Cas $t = \lambda y.t_0 \to_\beta \lambda y.t'_0 = t'$ avec $t_0 \to_\beta t'_0$ similaire, avec contexte $C = \lambda y.C_0$

λ-calcul pur : bilan
Formalisme minimal

- Variables
- Abstraction
- Application
- α-renommage
- β-réduction

En théorie, on n'a besoin de rien de plus ! *voir chapitre « λ-calculabilité »*

6 λ-calculs étendus

Programming with Computable Functions
Pour modéliser des langages de programmation, on peut étendre le λ-calcul

— entiers et arithmétique

— booléens, expressions conditionnelles

— structures de données

— fonctions récursives

— ...

On peut sélectionner les extensions qui nous intéressent *PCF* est un bouquet d'extensions standard

Extensions du λ-calcul
Ingrédients

— nouvelles formes syntaxiques

— règles de réduction

— extension des définitions (par ex. substitution)

— révision des preuves

Entiers et arithmétique
Nouvelles formes de termes

$$\begin{array}{rcll} t & ::= & \ldots & \\ & | & n & \text{constante entière} \\ & | & t_1\ op\ t_2 & \text{opération binaire } \oplus, \ominus, \ldots \end{array}$$

Nouvelles règles de base

$$n_1 \oplus n_2 \;\rightarrow\; n \qquad \text{avec} \quad n = n_1 + n_2$$

Nouvelles règles de congruence

$$\frac{t_1 \;\rightarrow\; t_1'}{t_1 \oplus t_2 \;\rightarrow\; t_1' \oplus t_2} \qquad \frac{t_2 \;\rightarrow\; t_2'}{t_1 \oplus t_2 \;\rightarrow\; t_1 \oplus t_2'}$$

Définitions étendues

$$\begin{array}{rcl} \mathsf{fv}(t_1\ op\ t_2) & = & \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2) \\ (t_1\ op\ t_2)\{x \leftarrow u\} & = & (t_1\{x \leftarrow u\})\ op\ (t_2\{x \leftarrow u\}) \end{array}$$

Booléens et expression conditionnelle
Nouvelles formes de termes

t	::=	…	
	|	T	vrai
	|	F	faux
	|	isZero(t)	test
	|	if t_1 then t_2 else t_3	conditionnelle

Nouvelles règles

isZero(0)	$\rightarrow$	T	
isZero(n)	$\rightarrow$	F	$n \neq 0$
if T then t_1 else t_2	$\rightarrow$	t_1	
if F then t_1 else t_2	$\rightarrow$	t_2	

+ règles de congruence

Paires
Nouvelles formes de termes

t	::=	…	
	|	$\langle t_1, t_2 \rangle$	paire
	|	$\pi_1(t)$	projection gauche
	|	$\pi_2(t)$	projection droite

Nouvelles règles

$\pi_1(\langle t_1, t_2 \rangle)$	$\rightarrow$	t_1
$\pi_2(\langle t_1, t_2 \rangle)$	$\rightarrow$	t_2

+ règles de congruence

Listes chaînées
Nouvelles formes de termes

t	::=	…	
	|	Nil	liste vide
	|	t_1::t_2	ajout d'un élément en tête
	|	isNil(t)	test
	|	hd(t)	tête
	|	tl(t)	queue

Nouvelles règles

isNil(Nil)	$\rightarrow$	T
isNil(t_1::t_2)	$\rightarrow$	F
hd(t_1::t_2)	$\rightarrow$	t_1
tl(t_1::t_2)	$\rightarrow$	t_2

+ règles de congruence

Récursion
Nouvelle forme de termes

$$\begin{array}{rcll} t & ::= & \ldots & \\ & | & \mathsf{Fix}(t) & \text{point fixe} \end{array}$$

Nouvelle règle

$$\mathsf{Fix}(t) \;\rightarrow\; t\ (\mathsf{Fix}(t))$$

+ règle de congruence

Exercice : réduction étendue
Calculer le résultat du terme suivant

$$\mathsf{Fix}(\ \lambda f s.\mathsf{if}\ \mathsf{isNil}(s)\ \mathsf{then}\ 0\ \mathsf{else}\ 1 \oplus (f(\mathsf{tl}(s)))\ )\ (2::4::8::\mathsf{Nil})$$

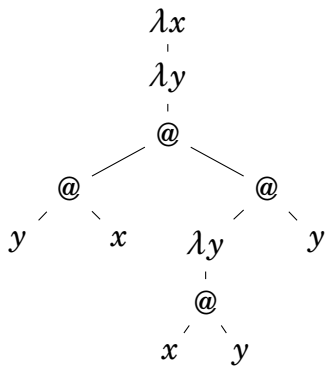
Correction On note $F = \lambda f s.\mathsf{if}\ \mathsf{isNil}(s)\ \mathsf{then}\ 0\ \mathsf{else}\ 1 \oplus (f(\mathsf{tl}(s)))$.

$$\begin{array}{ll} & \mathsf{Fix}(F)\ (2::4::8::\mathsf{Nil}) \\ \rightarrow & F\ (\mathsf{Fix}(F))\ (2::4::8::\mathsf{Nil}) \\ \rightarrow & (\lambda s.\mathsf{if}\ \mathsf{isNil}(s)\ \mathsf{then}\ 0\ \mathsf{else}\ 1 \oplus (\mathsf{Fix}(F))(\mathsf{tl}(s)))\ (2::4::8::\mathsf{Nil}) \\ \rightarrow & \mathsf{if}\ \mathsf{isNil}(2::4::8::\mathsf{Nil})\ \mathsf{then}\ 0\ \mathsf{else}\ 1 \oplus (\mathsf{Fix}(F))(\mathsf{tl}(2::4::8::\mathsf{Nil})) \\ \rightarrow & \mathsf{if}\ \mathsf{F}\ \mathsf{then}\ 0\ \mathsf{else}\ 1 \oplus (\mathsf{Fix}(F))(\mathsf{tl}(2::4::8::\mathsf{Nil})) \\ \rightarrow & 1 \oplus (\mathsf{Fix}(F))(\mathsf{tl}(2::4::8::\mathsf{Nil})) \\ \rightarrow & 1 \oplus (\mathsf{Fix}(F))(4::8::\mathsf{Nil}) \\ \ldots & \\ \rightarrow & 1 \oplus 1 \oplus 1 \oplus (\mathsf{Fix}(F)\ \mathsf{Nil}) \\ \rightarrow & 1 \oplus 1 \oplus 1 \oplus (F\ (\mathsf{Fix}(F))\ \mathsf{Nil}) \\ \rightarrow & 1 \oplus 1 \oplus 1 \oplus ((\lambda s.\mathsf{if}\ \mathsf{isNil}(s)\ \mathsf{then}\ 0\ \mathsf{else}\ 1 \oplus (\mathsf{Fix}(F))(\mathsf{tl}(s)))\ \mathsf{Nil}) \\ \rightarrow & 1 \oplus 1 \oplus 1 \oplus (\mathsf{if}\ \mathsf{isNil}(\mathsf{Nil})\ \mathsf{then}\ 0\ \mathsf{else}\ 1 \oplus (\mathsf{Fix}(F))(\mathsf{tl}(\mathsf{Nil}))) \\ \rightarrow & 1 \oplus 1 \oplus 1 \oplus 0 \\ \rightarrow & 1 \oplus 1 \oplus 1 \\ \rightarrow & 1 \oplus 2 \\ \rightarrow & 3 \end{array}$$

7 Notation de de Bruijn

Remplacer les variables par des numéros
Au lieu d'utiliser des noms de variables

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



on donne pour chaque occurrence de variable le nombre de λ à traverser pour retrouver son lieur

$$\lambda.\lambda.0\ 1\ ((\lambda.20)\ 0)$$

Avantage : plus besoin de gérer les renommages

Indices de de Bruijn, en caml

λ-termes avec indices de Bruijn

```
type terme =
  | Var of int
  | App of terme * terme
  | Abs of terme
```

Représentation du terme $\lambda.\lambda.0\ 1\ ((\lambda.20)\ 0)$

```
Abs(Abs(App(App(Var 0, Var 1),
            App(Abs(App(Var 2, Var 0)),
                Var 0))))
```

Substitutions et indices

β-réduction

— substitution des indices 0 relatifs au λ du radical

$$(\lambda.0\ (\lambda.0\ 1))\ t \quad \rightarrow_\beta \quad t\ (\lambda.0\ t)$$

— mais attention aux autres indices sous le λ du radical

$$(\lambda.0\ 1\ (\lambda.0\ 1))\ t \quad \not\rightarrow_\beta \quad t\ 1\ (\lambda.0\ t)$$

il faut les décrementer

— et attention de même aux indices de l'argument substitué

$$(\lambda.0\ 1\ (\lambda.0\ 1))\ 0 \quad \not\rightarrow_\beta \quad 0\ 1\ (\lambda.0\ 0)$$

il faut les incrémenter

Substitution en caml

Substitution de l'indice i

```
let rec subst t i u = match t with
  | Var j -> if i=j then u
       else if i<j then Var (j-1)
       else t
  | App(t1,t2) -> App(subst t1 i u,
                      subst t2 i u)
  | Abs t -> let u' = shift 0 u in
             Abs (subst t (i+1) u')
```

Fonction auxiliaire de décalage à partir de l'indice k

```
let rec shift k u = match u with
  | Var j -> if   k<=j
             then Var (j+1)
             else u
  | App(t1, t2) -> App(shift k t1,
                       shift k t2)
  | Abs t -> Abs (shift (k+1) t)
```

Exercice : notation de de Bruijn
Écrire les termes suivants en notation de de Bruijn

1. $\lambda x.(\lambda x.xy)(\lambda y.xy)$
2. $\lambda xy.x(\lambda y.(\lambda y.y)yz)$

Écrire le terme suivant en notation de de Bruijn, et le réduire

$$(\lambda f.f\ f)\ (\lambda ab.b\ a\ b)$$

Correction

1. $\lambda.(\lambda.02)(\lambda.10)$
2. $\lambda.\lambda.1(\lambda.(\lambda.0)03$
3. $$\begin{aligned}(\lambda.00)\ (\lambda.\lambda.010) &\rightarrow (\lambda.\lambda.010)\ (\lambda.\lambda.010)\\ &\rightarrow \lambda.0(\lambda.\lambda.010)0\end{aligned}$$

DM – *à rédiger et à me renvoyer avant le prochain cours*
Démonstration
Si $x \neq y$ et $x \notin \mathrm{fv}(v)$ alors

$$t\{x \leftarrow u\}\{y \leftarrow v\} \quad = \quad t\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\}$$