MI030 — APS
Analyse des programmes et sémantique

© Jacques Malenfant, 2010–2014

avec la participation initiale d'**Olena Rogovchenko**

Université Pierre et Marie Curie
UFR 919 Ingénierie
Jacques.Malenfant@lip6.fr

Cours 5
Le λ-calcul

Pourquoi étudier le λ-calcul ?

- Fondement des langages fonctionnels.
- Outil de base de la sémantique dénotationnelle pour représenter les fonctions mathématiques.

4 variétés de λ-expression

1. Variables (lettres minuscules)
2. Constantes prédéfinies (valeurs, opérateurs, ..., *i.e.*, les δ-*règles*)
3. Application de fonctions (combinateurs)
4. λ-abstractions (définitions de fonctions)

Grammaire des λ-expressions :

$$e \ ::= \ v \mid c \mid (e_1 \ e_2) \mid \lambda v.e$$

qui sont aussi appelées des λ-*termes*.

Quelques définitions

- *Variable liée* : la variable v dans $\lambda v.e$
- *Corps de fonction* : le terme e dans $\lambda v.e$
- *Opérateur* : le terme e_1 dans $(\underline{e_1} \ e_2)$
- *Opérande* : le terme e_2 dans $(e_1 \ \underline{e_2})$

Exemples de fonctions

$\lambda x.x$	fonction identité (polymorphique)	$(\lambda x.x \ 5) \Rightarrow 5$
$\lambda x.x+1$	fonction successeur	$((\lambda x.x+1) \ 5) \Rightarrow 6$
$\lambda f.\lambda x.(f \ (f \ x))$	fonction double	$(((\lambda f.\lambda x.(f \ (f \ x))) \ (\lambda x.x+1)) \ 5) \Rightarrow 7$

où $+$ est une constante fonctionnelle dont l'évaluation est définie par des δ-règles.

Plus de définitions ; notations

1. *Lettres majuscules* : méta-variables représentant des λ-expressions.
2. *Associativité à gauche* : $E_1 \ E_2 \ E_3 \ \equiv \ ((E_1 \ E_2) \ E_3)$
3. *Portée maximale à droite des définitions de fonctions* :
   $\lambda x.E_1 \ E_2 \ E_3 \ \equiv \ \lambda x.(E_1 \ E_2 \ E_3)$
   et non $(\lambda x.E_1) \ E_2 \ E_3$
4. *Curryfication des fonctions à plusieurs arguments* :
   $\lambda x \, y \, z.e \ \equiv \ \lambda x.\lambda y.\lambda z.e$
   mais $\lambda(x \, y \, z).e$ prend un argument qui est un triplet.
5. *Nommage des fonctions, mais par simple substitution textuelle (hors du λ-calcul)* : *Twice* $= \lambda f.\lambda x.(f \ (f \ x))$
   $(\mathit{Twice} \ (\lambda x.x+1) \ 5) \Rightarrow ((\lambda f.\lambda x.(f \ (f \ x))) \ (\lambda x.x+1) \ 5) \Rightarrow 7$

Fonctions curryfiées

Ainsi nommées selon le mathématicien Curry qui les a étudiées la première fois.

Soit `sum(a, b) = a + b` une fonction à représenter en λ-calcul, deux représentations sont possibles :

- $\lambda(a,b).a+b$ de type $int \times int \rightarrow int$
- $\lambda a.\lambda b.a+b$ de type $int \rightarrow int \rightarrow int$ (version curryfiée)

Le résultat de $((\lambda a.\lambda b.a+b)\ 5) \Rightarrow \lambda b.5+b$ est alors vu comme la fonction qui ajoute 5 à son argument.

Mode d'évaluation

- Évaluation, opérationnellement fondée sur la *réécriture* ou la *réduction* des λ-termes.
- Opération fondamentale : substitution des variables libres par des expressions dans une application pour réaliser le passage des paramètres.
- Exemple : $((\lambda a.\lambda b.a+b)\ 5) \Rightarrow \lambda b.5+b$
  on a substitué la variable a par la valeur 5 dans le corps de la fonction.
- Pour les opérations « hors λ-calcul », comme l'addition, elles sont gérées par des règles de réduction spécifiques appelées δ-*règles*.

Substitution et capture de variables I

Définition (occurrence libre, liée)

Un occurrence d'une variable v dans une λ-expression est dite *liée* si elle est dans la portée d'un λv, sinon elle est dite *libre*.

Notation ($E[v \rightarrow E_1]$)

La notation $E[v \rightarrow E_1]$ représente la substitution de toutes les occurrences libres de la variable v dans la λ-expression E par la λ-expression E_1.

- Attention cependant aux captures de variables !

Substitution et capture de variables II

- S'il y a capture, la subsitution *n'est pas valide*, et alors il faut d'abord renommer les variables susceptibles d'être capturées :
  $(\lambda x.v)[v \rightarrow \lambda y.y+x] \;\Rightarrow\; \lambda x.\lambda y.y+x$ ⚠ capture de x !
  $(\lambda x.v)[v \rightarrow \lambda y.y+z] \;\Rightarrow\; \lambda x.\lambda y.y+z$ OK, pas de capture...

Calcul de l'ensemble des variables libres

Définition (variables libres)

L'ensemble des *variables libres* dans une λ-expression e, noté $VL(e)$ est défini par :

- $VL(c) = \emptyset$ $\quad c \in constante$
- $VL(x) = \{x\}$ $\quad$ pour toute variable x
- $VL(E_1\ E_2) = VL(E_1) \cup VL(E_2)$
- $VL(\lambda x.e) = VL(e) - \{x\}$

Une λ-expression e telle que $VL(e) = \emptyset$ est dite *close*.

Nota : c'est une définition dirigée par la syntaxe !

Calcul des substitutions

Définition (substitution)

La substitution $E[v \rightarrow E_1]$ est définie par :

- $v[v \rightarrow E_1] = E_1$
- $x[v \rightarrow E_1] = x$ $\quad$ si $x \neq v$
- $c[v \rightarrow E_1] = c$ $\quad$ si c est une constante
- $(E\ E')[v \rightarrow E_1] = (E[v \rightarrow E_1]\ E'[v \rightarrow E_1])$
- $(\lambda v.E)[v \rightarrow E_1] = \lambda v.E$
- $(\lambda x.E)[v \rightarrow E_1] = \lambda x.(E[v \rightarrow E_1])$ $\quad$ si $x \neq v$ et $x \notin VL(E_1)$
- $(\lambda x.E)[v \rightarrow E_1] = \lambda z.(E[x \rightarrow z][v \rightarrow E_1])$
  si $x \neq v$ et $x \in VL(E_1)$ et $z \neq x$ et $z \notin VL(E_1)$

Exemple de calcul d'une substitution

$$
\begin{aligned}
(\lambda y.(\lambda f.f\ x)\ y)[x \rightarrow f\ y] &= (\lambda y.(\lambda f.f\ x)\ y)[y \rightarrow z][x \rightarrow f\ y] \\
&= (\lambda z.(\lambda f.f\ x)\ z)[x \rightarrow f\ y] \\
&= \lambda z.((\lambda f.f\ x)[x \rightarrow f\ y])\ (z[x \rightarrow f\ y]) \\
&= \lambda z.((\lambda f.f\ x)[x \rightarrow f\ y])\ z \\
&= \lambda z.((\lambda f.f\ x)[f \rightarrow g][x \rightarrow f\ y])\ z \\
&= \lambda z.((\lambda g.g\ x)[x \rightarrow f\ y])\ z \\
&= \lambda z.(\lambda g.g\ x[x \rightarrow f\ y])\ z \\
&= \lambda z.(\lambda g.g\ (f\ y))\ z
\end{aligned}
$$

La réduction des λ-expressions I

- Évaluation $\equiv$ simplification : réduire une expression jusqu'à ce que plus aucune règle ne s'applique.
- Règle principale : β-réduction pour l'application de fonction.
- Seconde règle : α-conversion pour le renommage des variables pour éviter les captures.

La réduction des λ-expressions II

Définition (α-conversion)

Si v et w sont des variables et E une λ-expression, alors l'*alpha*-conversion est définie par :

$$\lambda v.E \Rightarrow_\alpha \lambda w.E[v \rightarrow w]$$

si w n'apparaît pas dans E.

Les termes $\lambda v.E$ et $\lambda w.E[v \rightarrow w]$ sont dits α-*congruents*.

La réduction des λ-expressions III

Définition (β-réduction)

Si v est une variable et E ; E_1 des λ-expressions, alors la β-réduction est définie par :

$$(\lambda v.E)\ E_1 \Rightarrow_\beta E[v \rightarrow E_1]$$

Le terme $(\lambda v.E)\ E_1$ est appelé β-redex.

L'évaluation d'une λ-expression est constituée d'une suite de β-réductions, possiblement entrelacée d'α-conversion.

Elle est notée $\Rightarrow_\beta^*$

Les δ-règles I

- Le λ-calcul est très minimal, en n'admettant que la β-réduction comme règle de réduction.
- Comment représenter les calculs de base, comme les entiers naturels et l'addition ?
- Il existe un encodage des entiers naturels sous la forme de λ-termes et de l'addition comme λ-expression, mais son intérêt est purement théorique.
- En pratique, on trouve plus commode d'étendre le λ-calcul pour inclure en quelque sorte des types de données élémentaires et leurs opérations sous la forme de constantes.

Les δ-règles II

- Ces opérations n'étant pas formulées comme des fonctions du λ-calcul, leur « traitement » est délégué à des règles de réductions spécifiques ajoutées à la β-réduction.
- Exemple :
  Constantes : valeurs dans $\mathbb{N}$ et l'opération *add*.
  δ-règles :
  $$(add\ \Lambda_1\ \Lambda_2) \rightarrow_\delta \Lambda_1 + \Lambda_2 \text{ si } \Lambda_1, \Lambda_2 \in \mathbb{N}$$
  où '+' représente l'addition sur les entiers naturels.

Évaluation des λ-expressions I

But de l'évaluation : réduire la λ-expression à une forme la plus simple possible, et considérer le λ-terme résultant comme la valeur de l'expression.

Définition (forme normale)

Une λ-expression est dite en *forme normale* si elle ne contient aucun β-redex (ni aucune δ-règle applicable), si bien qu'elle ne peut être réduite davantage par β-réduction (ou application d'une δ-règle).

Évaluation des λ-expressions II

Quatre questions fondamentales :

1. Est-ce que toute λ-expression peut être réduite à une forme normale ?
2. Existe-t'il plus d'une façon (séquences de réductions) de réduire une λ-expression ?
3. S'il existe plus d'une façon de réduire une λ-expression, donnent-elles toutes la même forme normale ?
4. Existe-t'il une façon de réduire les λ-expression qui garantisse l'obtention d'une forme normale ?

Réponses aux questions... I

Avant d'aller plus avant, donnons quelques réponses informelles aux quatre questions précédentes :

1. Non, toutes les λ-expressions ne sont pas réductibles à une forme normale. Exemple :

$$((\lambda x.x\ x)\ (\lambda x.x\ x)) \ \Rightarrow_\beta \ ((\lambda x.x\ x)\ (\lambda x.x\ x)) \ \Rightarrow_\beta \ \ldots$$

Réponses aux questions... II

2. Oui, il existe souvent plusieurs façons de réduire une même λ-expression. Par exemple, l'expression $(((\lambda x.\lambda y.y + ((\lambda z.x \times z)\ 3))\ 7)\ 5)$ possède les deux séquences de réduction suivantes :

$$\begin{aligned}
(((\lambda x.\lambda y.y + ((\lambda z.x \times z)\ 3))\ 7)\ 5) &\Rightarrow_\beta ((\lambda y.y + ((\lambda z.7 \times z)\ 3))\ 5)\\
&\Rightarrow_\beta 5 + ((\lambda z.7 \times z)\ 3)\\
&\Rightarrow_\beta 5 + 7 \times 3)\\
&\Rightarrow_\beta 5 + 21)\\
&\Rightarrow_\beta 26
\end{aligned}$$

Réponses aux questions... III

ou encore :

$$\begin{aligned}
(((\lambda x.\lambda y.y + ((\lambda z.x \times z)\ 3))\ 7)\ 5) &\Rightarrow_\beta (((\lambda x.\lambda y.y + (x \times 3)\ 7)\ 5)\\
&\Rightarrow_\beta ((\lambda y.y + (7 \times 3))\ 5)\\
&\Rightarrow_\beta ((\lambda y.y + 21)\ 5)\\
&\Rightarrow_\beta (5 + 21)\\
&\Rightarrow_\beta 26
\end{aligned}$$

3. Lorsqu'elles arrivent à donner une forme normale, toutes les stratégies de réduction donnent la même.

Réponses aux questions... IV

4. Oui, il existe une stratégie de réduction qui trouve toujours la forme normale, mais elle a aussi ses inconvénients si on veut l'implanter sur ordinateur.

Stratégie de réduction I

- Par « façons de réduire » une λ-expression, on se réfère au fait qu'il peut exister plus d'un β-redex dans l'expression, et alors l'ordre dans lequel on réduit ces redex suscite les questions précédentes.

Définition (stratégie de réduction)

Une *stratégie de réduction* définit un ordre dans lequel les β-redex sont réduits pour tenter d'obtenir une forme normale.

- Les deux stratégies de réduction les plus connues sont :

Stratégie de réduction II

ordre applicatif : réduit toujours le β-redex le plus à gauche et le plus « profond » dans l'arborescence formée par le terme.

ordre normal : réduit toujours le β-redex le plus à gauche et le plus « extérieur » dans l'arborescence formée par le terme.

- L'ordre applicatif doit son nom au fait qu'il correspond à l'ordre d'évaluation courant des appels de fonction : on évalue d'abord les arguments (de gauche à droite) qui sont des sous-termes (plus profond) du terme application, puis on appelle la fonction, c'est-à-dire qu'on fait la β-réduction du terme applicatif lui-même.

Stratégie de réduction III

- L'ordre normal, pour sa part, effectue d'abord la β-réduction (réduction de l'application) et donc passe les arguments sans les évaluer (termes plus profond). Si un paramètre de la fonction est utilisé à plusieurs endroits dans son corps, cet ordre d'évaluation va donc le répéter à chaque occurrence, ce qui demandera donc une implantation plus astucieuse si on veut éviter de le réduire plusieurs fois pendant l'évaluation du corps de la fonction.

Théorème de Church-Rosser I

Theorème (Church-Rosser I)

Pour toutes λ-expressions E, F, G, si $E \Rightarrow^* F$ et $E \Rightarrow^* G$, alors il existe une λ-expressions Z telle que $F \Rightarrow^* Z$ et $G \Rightarrow^* Z$.

C'est la propriété dite de confluence, du *losange* ou encore du *diamant* :

$$\begin{tikzcd} & E \arrow[dl, Rightarrow, "*"'] \arrow[dr, Rightarrow, "*"] & \\ F \arrow[dr, Rightarrow, "*"'] & & G \arrow[dl, Rightarrow, "*"] \\ & Z & \end{tikzcd}$$

Corollaire

Corollaire

Pour toutes λ-expressions E, F, G, si $E \Rightarrow^* F$ et $E \Rightarrow^* G$, et si F et G sont des formes normales, alors F et G sont des variantes du même λ-terme à des α-conversions près.

Théorème de Church-Rosser II

Théorème (Church-Rosser II)

Pour toutes λ-expressions E et N où N est une forme normale, si $E \Rightarrow^* N$ alors il existe une réduction selon la stratégie en ordre normal de E à N.

Retour sur la quatrième question...

- Ceci répond plus complètement à la quatrième question posée plus tôt : l'existence d'une stratégie de réduction qui trouve toujours la forme normale si elle existe.
- En fait, en conjonction avec la réponse à la première question qui montre qu'il existe des termes qui n'ont pas de forme normale, la stratégie de réduction en ordre normal soit trouve la forme normale si elle existe, soit ne s'arrête pas, c'est-à-dire peut continuer indéfiniment à appliquer la β-réduction à des termes qui n'ont pas de forme normale, comme celui montré à la question 1.

Thèse de Church

Thèse de Church

toutes les fonctions calculables sur les entiers sont celles définissables par le λ-calcul pur et correspondent aux fonctions calculables par une machine de Turing.

Rappelons que Turing a démontré l'indécidabilité du problème d'arrêt...

Relation avec le passage de paramètres

Il existe donc, comme mentionné précédemment, une relation étroite entre stratégie de réduction et mode de passage de paramètres aux fonctions :

- Appel par *nom* : réduction en ordre normal, sauf qu'aucun redex apparaissant dans une abstraction n'est jamais réduit.
- Appel par *valeur* : réduction en ordre applicatif, sauf qu'aucun redex apparaissant dans une abstraction n'est jamais réduit.

Comment écrire une fonction récursive en λ-calcul I

- En λ-calcul, les fonctions n'ont pas de nom, elles ne peuvent donc pas se désigner elle-même de manière à s'appeler récursivement.
- On a vu une notation pour donner des noms à des λ-expressions ($Twice = \ldots$), mais
  - c'est une notation qui est « hors » du λ-calcul, et
  - elle suppose qu'on peut faire une substitution textuelle, ce qui n'est pas possible si le terme se mentionne lui-même dans son corps.

Comment écrire une fonction récursive en λ-calcul II

- Le λ-calcul utilise donc plutôt un terme particulier qui permet, lors de sa réduction, de réappliquer une fonction à l'intérieur de son corps autant de fois que nécessaire.
- C'est le *combinateur de point fixe* !

Théorème du point fixe I

Théorème (point fixe)

Pour tout λ-terme F, il existe un λ-terme X tel que $F\ X = X$. X est appelé *point fixe* de F.

Preuve. Soit $W \equiv \lambda x.F\ (x\ x)$ et $X \equiv W\ W$. Alors

$$X \equiv W\ W \equiv (\lambda x.F\ (x\ x))\ W \Rightarrow_\beta F\ (W\ W) \equiv F\ X$$

- Les points fixes nous sont familiers. Par exemple, 1 est le point fixe de la fonction racine carrée, car $\sqrt{1} = 1$.

Syntaxe du λ-calcul
Sémantique opérationnelle des λ-expressions
Stratégies de réduction
Point fixe et fonctions récursives

Théorème du point fixe II

- En λ-calcul, la fonction identité est son propre point fixe :

$$(\lambda x.x)\ (\lambda x.x) \Rightarrow_\beta (\lambda x.x)$$

- Il est plus surprenant d'apprendre que *tout terme* a un point fixe !
- De fait, le théorème du point fixe possède une preuve constructive, c'est-à-dire que la preuve exhibe le terme qui est le point fixe, et ainsi fournit une *recette* pour le construire le point fixe de tout terme. Pour $F \equiv \lambda x.\lambda y.(x\ y)$, il suffit d'utiliser le terme W et d'y substituer F :

$$W \equiv \lambda z.F\ (z\ z) \equiv \lambda z.((\lambda x.\lambda y.(x\ y))\ (z\ z)) \equiv \lambda z.\lambda y.((z\ z)\ y)$$

Syntaxe du λ-calcul
Sémantique opérationnelle des λ-expressions
Stratégies de réduction
Point fixe et fonctions récursives

Théorème du point fixe III

Le terme $X \equiv W\ W \equiv ((\lambda z.\lambda y.((z\ z)\ y))\ (\lambda z.\lambda y.((z\ z)\ y)))$ est le point fixe cherché. En effet,

$$\begin{aligned} X &\equiv (W\ W) \\ &\equiv ((\lambda z.\lambda y.((z\ z)\ y))\ W) \\ &\Rightarrow_\beta \lambda y.((W\ W)\ y) \\ &\Rightarrow_\beta ((\lambda x.\lambda y.(x\ y))\ (W\ W)) \\ &\equiv (F\ (W\ W)) \\ &\equiv F\ X \end{aligned}$$

Syntaxe du λ-calcul
Sémantique opérationnelle des λ-expressions
Stratégies de réduction
Point fixe et fonctions récursives

Un combinateur de point fixe

- Le théorème du point fixe nous dit qu'on peut trouver le point fixe de tout terme F en substituant ce terme à l'intérieur du terme $W\ W$.
- Cette observation inspire la définition suivante qui applique simplement le λ-calcul à ce procédé.

Définition (combinateur de point fixe)

$$Y \equiv \lambda f.((\lambda x.(f\ (x\ x)))\ (\lambda x.(f\ (x\ x))))$$

- Notez que pour tout terme F, l'expression $Y\ F$ va donner son point fixe, d'où le nom de combinateur de point fixe pour Y.

Combinateur de point fixe et fonction récursive I

- Considérons la fonction factorielle que nous pourrions définir dans un langage de programmation quelconque sous la forme :

```
fact n := if n = 0 then 1 else n * (fact n-1)
```

- Comme nous l'avons vu, il serait tentant d'écrire :

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

mais ce n'est pas un λ-terme...

Combinateur de point fixe et fonction récursive II

- Notons cependant que nous pourrions faire apparaître le terme *Fact* dans le λ-terme comme un paramètre d'une λ-expression :

$$\mathit{Fact} = \lambda f.\lambda n.\mathit{if}\ (=\ n\ 0)\ \mathit{then}\ 1\ \mathit{else}\ (\times\ n\ (f\ (-\ n\ 1)))$$

et la récursivité pourrait alors être obtenue en appliquant ce terme sur lui-même !

- C'est exactement ce que le combinateur de point fixe va permettre. Calculons $T = ((Y\ \mathit{Fact})\ 2)$:

Combinateur de point fixe et fonction récursive III

$$T \equiv ((Y\ \mathit{Fact})\ 2)$$
$$\equiv ((\lambda f.((\lambda x.(f\ (x\ x)))\ (\lambda x.(f\ (x\ x))))\ \mathit{Fact})\ 2)$$
$$\Rightarrow_\beta (((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x))))\ 2)$$
$$\Rightarrow_\beta ((\mathit{Fact}\ ((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x)))))\ 2)$$
$$\Rightarrow_\beta ((\lambda n.\mathit{if}\ (=\ n\ 0)\ \mathit{then}\ 1\ \mathit{else}\ (\times\ n\ (((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x))))\ (-\ n\ 1))))\ 2)$$
$$\Rightarrow_\beta \mathit{if}\ (=\ 2\ 0)\ \mathit{then}\ 1\ \mathit{else}\ (\times\ 2\ (((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x))))\ (-\ 2\ 1)))$$
$$\Rightarrow_\delta (\times\ 2\ (((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x))))\ (-\ 2\ 1)))$$
$$\Rightarrow_\delta (\times\ 2\ (((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x))))\ 1))$$
$$\Rightarrow_\beta (\times\ 2\ ((\mathit{Fact}\ ((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x)))))\ 1))$$
$$\Rightarrow_\beta (\times\ 2\ ((\lambda n.\mathit{if}\ (=\ n\ 0)\ \mathit{then}\ 1\ \mathit{else}\ (\times\ n\ (((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x))))\ (-\ n\ 1)))\ 1))$$
$$\Rightarrow_\beta (\times\ 2\ (\mathit{if}\ (=\ 1\ 0)\ \mathit{then}\ 1\ \mathit{else}\ (\times\ 1\ (((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x))))\ (-\ 1\ 1)))))$$

Combinateur de point fixe et fonction récursive IV

$$\Rightarrow_\delta (\times\ 2\ (\times\ 1\ (((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x))))\ (-\ 1\ 1))))$$
$$\Rightarrow_\delta (\times\ 2\ (\times\ 1\ (((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x))))\ 0)))$$
$$\Rightarrow_\beta (\times\ 2\ (\times\ 1\ ((\mathit{Fact}\ ((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x)))))\ 0)))$$
$$\Rightarrow_\beta (\times\ 2\ (\times\ 1\ ((\lambda n.\mathit{if}\ (=\ n\ 0)\ \mathit{then}\ 1\ \mathit{else}\ (\times\ n\ (((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x))))\ (-\ n\ 1))))\ 0)))$$
$$\Rightarrow_\delta (\times\ 2\ (\times\ 1\ (\mathit{if}\ (=\ 0\ 0)\ \mathit{then}\ 1\ \mathit{else}\ (\times\ 0\ (((\lambda x.(\mathit{Fact}\ (x\ x)))\ (\lambda x.(\mathit{Fact}\ (x\ x))))\ (-\ 0\ 1))))))$$
$$\Rightarrow_\delta (\times\ 2\ (\times\ 1\ 1)) \Rightarrow_\delta (\times\ 2\ 1) \Rightarrow_\delta 2$$

- Notons que l'obtention de la forme normale dans cette réduction dépend de manière cruciale sur l'utilisation de l'ordre normal dans les β-réductions, sinon il y aurait divergence (essayez !).
- Cette observation sera importante quand il s'agira d'implanter le λ-calcul sur ordinateur.

Activités complémentaires

- Regarder le tutoriel de Barendregt et Barendsen sur le λ-calcul.