La semaine prochaine, **TD** *et pas* **TP**
(en salles B2 – le matin – et **A2** *– l'après-midi –)*

Un peu de λ-calcul

définition, stratégies

Les *termes* du λ-calcul

la syntaxe:

$M \quad ::= \lambda x.\, M \mid M\, M' \mid x$ `fun x -> m | m m' | x`

- $\lambda x.\, M$: abstraction $M\, M'$: application
- λ est un lieur

  $\mathrm{vl}(M)$: variables libres de M
- α-conversion: on identifie $\lambda x.\, M$ et $\lambda y.\, M[y/x]$, pour peu que $y \notin \mathrm{vl}(M)$.
- exemples: $\lambda x.\, x$ $\lambda xy.\, (y\, x)$ $\lambda u.\, v\, (\lambda u.\, u)\, u$
  - on note $\lambda xyz.\, M$ pour $\lambda x.\, \lambda y.\, \lambda z.\, M$ *(cf.* `fun x y z -> ..`*)*
  - similairement, $M\, M'\, M''$ représente $(M\, M')\, M''$
- un terme du λ-calcul peut être vu comme un arbre (de syntaxe)

Un calcul de *fonctions*

règle de calcul: la β-réduction

$(\lambda x.\, M)\; M' \quad \to \quad M[M'/x]$

- exemple: $(\lambda uv.\, (v\, u\, u))\; (\lambda x.\, x) \quad \to \quad \lambda v.\, (v\, (\lambda x.\, x)\, (\lambda x.\, x))$
- $M[M'/x]$: on remplace x par M' dans M *en faisant des α-conversions si nécessaire* *(substitution)*
- le calcul $=$ passer son argument à une fonction
  notion de fonction différente de la notion usuelle en maths
- $(\lambda x.\, M)\, M'$ est un *redex*

  un terme peut "contenir" plusieurs redex

  qui sont des sous-arbres de la forme

Relation de réduction

- on définit la manière dont les termes se réduisent par des *règles d'inférence*

$$\frac{}{(\lambda x.\, M)\, M' \;\to\; M[M'/x]}\;\beta \qquad \frac{M \to M'}{M\,N \to M'\,N}\;\mu_l$$

$$\frac{M \to M'}{N\,M \to N\,M'}\;\mu_r \qquad \frac{M \to M'}{\lambda x.\, M \to \lambda x.\, M'}\;\xi$$

- β, μ_l, μ_r, ξ sont les noms des règles (pour faire joli)
- implicitement, chaque "identificateur" (*metavariable*) qu'on mentionne est quantifié universellement

  pour tout M, pour tout N, pour tout x,...
- au-dessus: prémisses/hypothèses, en dessous: la conclusion
- ces règles définissent des *arbres de dérivation*
  - aux feuilles: β / aux nœuds: μ_l, μ_r, ξ
- ceci définit la relation de réduction, notée $\to$

  c'est en fait le même genre de définition que pour la syntaxe
- allons-y avec $((\lambda xyz.\, y\,x)\,\lambda u.\, u)\;((\lambda h.\, h\,h)\,(\lambda t.\, t))$

Réduction: propriétés

- *confluence*
  - dans un terme donné, un certain nombre de redex $M \nearrow \to \searrow$
  - un choix risque-t-il d'être irrémédiable?
    réponse: non, le λ-calcul est confluent
- *terminaison*
  - réduisons $\Omega \;=\; (\lambda x.\,(x\,x))\;(\lambda x.\,(x\,x))$ (qui est un redex)

    NB: quand on écrit $\Omega \to \Omega$, on est informel
  - certains termes du λ-calcul divergent

    noter au passage qu'on utilise deux fois l'argument x, et aussi qu'il y a une auto-application
- ...ça plane, allons vers quelque chose qui soit plus proche d'un langage de programmation

Le λ-calcul *faible*

$$\frac{}{(\lambda x.\, M)\, M' \;\to\; M[M'/x]}\;\beta \qquad \frac{M \to M'}{M\,N \to M'\,N}\;\mu_l \qquad \frac{M \to M'}{N\,M \to N\,M'}\;\mu_r$$

$$\frac{M \to M'}{\lambda x.\, M \to \lambda x.\, M'}\;\xi$$

- en Caml, la règle ξ n'est pas autorisée: λ-calcul *faible*
- ceci définit *une autre* notion de réduction des termes
  - ainsi on peut compiler les fonctions

    "si on veut pouvoir réduire sous λ, il faut disposer du source"
  - en λ faible, toute fonction est une *valeur* (i.e., quelque chose qu'on ne peut plus réduire)
    - même $\lambda y.\, \Omega$, où $\Omega \;=\; (\lambda x.\,(x\,x))\;(\lambda x.\,(x\,x))$
    - $\lambda y.\, \Omega$ diverge pour le λ-calcul, et converge pour le λ-calcul faible
- en virant la règle ξ, est-on arrivé à Caml?
- **non**, car il faut encore spécifier comment choisir en μ_l et μ_r

La stratégie outermost

on élimine ξ et μ_r: $\dfrac{}{(\lambda x.\, M)\, M' \;\to\; M[M'/x]}\;\beta$ $\dfrac{M \to M'}{M\,N \to M'\,N}\;\mu_l$

- on va chercher le redex le plus à gauche, et on réduit (β)
- intuitivement, lorsqu'on réduit $M_1\ M_2\ M_3$
  - on attaque M_1 jusqu'à ce que ça devienne une fonction
  - on passe alors l'argument M_2 (tel quel, non réduit), et on recommence jusqu'à ce que ça devienne une fonction
  - on passe M_3, et on recommence
- $\lambda\ (\beta, \xi, \mu_l, \mu_r) \supset \lambda$ faible $(\beta, \mu_l, \mu_r) \supset \lambda$ outermost (β, μ_l)
- rappel: outermost est la stratégie des paresseux (outermost + partage = stratégie de Haskell)
- en Caml, un argument est évalué avant d'être passé à une fonction

  face à un terme de la forme $M_1\,M_2$, on veut faire "un peu de μ_l et un peu de μ_r"

La stratégie innermost

- on ne passe à une fonction qu'un argument qu'on ne peut plus réduire (une *valeur*)

  valeurs $V ::= \lambda x.\, M \mid x$

  autrement dit, un terme est une valeur si c'est une variable ou une abstraction

$$\frac{}{(\lambda x.\, M)\, V \;\to\; M[V/x]}\ \beta_v$$

$$\frac{M \to M'}{M\,N \to M'\,N}\ \mu_l \qquad \frac{M \to M'}{M\,V \to M'\,V}\ \mu_{lv}$$

$$\frac{M \to M'}{N\,M \to N\,M'}\ \mu_r \qquad \frac{M \to M'}{V\,M \to V\,M'}\ \mu_{rv} \qquad \frac{M \to M'}{N\,M \to N\,M'}\ \mu_r$$

- on y est presque: il faut encore 'diriger' le choix entre μ_l et μ_r