Un peu de λ-calcul

Qu'est-ce?

- le cœur d'un langage de programmation fonctionnelle
  - étudier une extension nouvelle pour un langage de programmation passe volontiers par l'étude d'un λ-calcul étendu par *quelques* mécanismes
- le formalisme dans lequel sont développées de nombreuses méthodes pour l'analyse de programmes
  - systèmes de types
  - preuve de programmes
- avant cela:
  un modèle élémentaire exprimant la notion de calcul

Les *termes* du λ-calcul

la syntaxe:

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

$\lambda x.M$, **c'est** `fun x -> M`

- $\lambda x.\, M$: abstraction $M\, M'$: application
- il n'y a plus que des fonctions
  pas d'`int`, de `bool`, de types somme, de filtrage, de `ref`, d'exceptions
- 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 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)*
- exemple: les entiers de Church
  $1 = \lambda f\, x.\, (f\, x) \quad 2 = \lambda f\, x.\, (f\, (f\, x)) \quad n = \lambda f\, x.\, (f^n\, x)$
  - définition du successeur? de l'addition? de la multiplication?
    $\lambda n.\, \lambda f\, x.\, f\, (n\, f\, x) \quad \lambda n\, m.\, \lambda f\, x.\, (n\, f\, (m\, f\, x)) \quad \lambda n\, m.\, \lambda f\, x.\, (n\, (m\, f)\, x)$
  - définition de zéro? $\lambda f\, x.\, x$
  - type des entiers? `('a -> 'a) -> 'a -> 'a`
- *calculer* $\leftrightarrow$ 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

Relation de réduction

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

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

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

- 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

la semaine prochaine, un **TD** *et non un TP*

surveillez le panneau d'affichage pour savoir où

Le λ-calcul

- termes du λ-calcul: $M \ ::= \ \lambda x.\, M \mid M\, M' \mid x$
- calcul dans les termes: β-reduction

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

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

- un axiome (β), trois règles d'inférence
- un terme *est* un β-redex, ou *contient* un β-redex

Exemples de dérivations pour la réduction

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

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

réductions de $((\lambda xyz.\, y\, x)\, \lambda u.\, u)\ ((\lambda h.\, h\, h)\, \lambda t.\, t)$

$$\frac{\dfrac{}{(\lambda h.\, h\, h)\, \lambda t.\, t \to (\lambda t.\, t)\, \lambda t'.\, t'}\ \beta}{((\lambda xyz.\, y\, x)\, \lambda u.\, u)\ ((\lambda h.\, h\, h)\, \lambda t.\, t) \to ((\lambda xyz.\, y\, x)\, \lambda u.\, u)\ ((\lambda t.\, t)\, \lambda t'.\, t')}\ \mu_r$$

$$\frac{\dfrac{}{(\lambda xyz.\, y\, x)\, \lambda u.\, u \to \lambda yz.\, y\, (\lambda u.\, u)}\ \beta}{((\lambda xyz.\, y\, x)\, \lambda u.\, u)\ ((\lambda h.\, h\, h)\, \lambda t.\, t) \to (\lambda yz.\, y\, (\lambda u.\, u))\ ((\lambda h.\, h\, h)\, \lambda t.\, t)}\ \mu_l$$

Des arbres, et encore des arbres

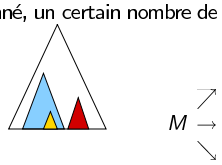
- on passe son temps à définir des arbres
- la syntaxe, c'est des arbres
  - un terme du λ-calcul est un arbre
  - un redex est un sous-arbre de la forme
- pour réduire un terme, on construit un *arbre de dérivation*
  - la question suivante
    "étant donnés deux termes M et N, est-ce que $M \to N$?"
    se reformule en
    "existe-t-il un arbre de dérivation ayant $M \to N$ à la racine?"
  - idem pour "quels sont les M' tels que $M \to M'$?"
  - similairement, pour Caml, on aurait des règles du genre

$$\frac{e_1 \to e_1' \qquad e_2 \to e_2'}{(e_1, e_2) \to (e_1', e_2')}$$

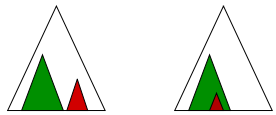
$$\frac{e_1 \to \texttt{true} \qquad e_2 \to e'}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \to e'} \qquad \frac{e_1 \to \texttt{false} \qquad e_3 \to e'}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \to e'}$$

Propriétés de la réduction: confluence

- dans un terme donné, un certain nombre de redex



- un choix risque-t-il d'être irrémédiable?



réponse: non, le λ-calcul est confluent

- mais on n'a pas donné de preuve:
  on a **esquissé** une preuve de *confluence locale* (cf. Prog 2)

Propriétés de la réduction: terminaison

- réduisons $\Omega \;=\; (\lambda x.\,(x\,x))\;(\lambda x.\,(x\,x))$ (qui est un redex)
- certains termes du λ-calcul divergent

  noter au passage qu'on utilise deux fois l'argument x, et aussi qu'il y a de l'auto-application
- types simples pour le λ-calcul:
  - on n'a droit à $(f\,x)$ que si $f : T \to T'$ et $x : T$
    (alors $(f\,x) : T'$)
  - Thm.: si M est bien typé, alors M ne diverge pas
  - existe-t-il des termes mal typés qui ne divergent pas?
- on a un *modèle de calcul*, pas mal abstrait
  - syntaxe, réduction ($M \to M'$)
  - en quoi est-on proche (ou pas) de Caml?
  - que faire pour voir le λ-calcul comme un *modèle de la programmation* (fonctionnelle)?

λ-calcul: stratégies

Le λ-calcul *faible*

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

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

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

- en Caml, la règle ξ n'est pas autorisée: λ-calcul *faible*
  - ainsi on peut *compiler* les fonctions
    "si on veut pouvoir réduire sous λ, il faut disposer du source"
- ceci définit *une autre* notion de réduction des termes: $\to_{\mathrm{f}}$
  - 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 $\quad \lambda y.\,\Omega \not\to_{\mathrm{f}}$
- une fois la règle ξ éliminée. sait-on "exécuter les

La stratégie innermost

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

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

  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

La stratégie outermost

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

- on va chercher le redex le plus à gauche (μ_l), 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)

  $(M \to_{\mathrm{o}} M')$ implique $(M \to_{\mathrm{f}} M')$, qui implique $(M \to M')$
- 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 $M_1\,M_2$, on veut faire "du μ_l et du μ_r"