Fondements de la programmation fonctionnelle

Lionel Vaux Auclair

I2M, université d'Aix-Marseille

Logique HUGO@Polytech, 2021–2022

λ-termes

L'ensemble Λ des λ-termes est défini inductivement par :

$$\Lambda \ni s, t, \ldots ::= x \mid \lambda x.s \mid s\, t$$

où x parcourt un ensemble infini dénombrable de variables $\mathcal{V}$
et les occurrences de x dans s sont liées par l'abstraction $\lambda x.s$

Il faut voir ces termes comme *codant* des fonctions :

- $\lambda x.s$ abstrait la variable x dans s : c'est le code de $x \mapsto s$;

  Ne pas confondre une expression dépendant d'une variable avec la fonction correspondante (par exemple, $x + 1 \neq y + 1$, tandis que $(x \mapsto x + 1) = (y \mapsto y + 1)$).
- $s\, t$ est l'application du terme s au terme t ;
- le calcul fait le lien entre les deux.

β-réduction

L'expression $(\lambda x.s)\, t$ se réduit en $s[x := t]$ (la valeur de s pour $x := t$).

β-réduction

L'expression $(\lambda x.s)\, t$ se réduit en $s[x := t]$ (la valeur de s pour $x := t$).

Plus formellement : on note $s[x := t]$ le résultat de la substitution de t à x dans s.

Une analogie : si P est un polynôme en x et y, $x \mapsto P$ associe à chaque valeur de x le polynôme en y obtenu en remplaçant x par sa valeur.

Exemples

Quelques termes :

$$
\begin{aligned}
I &:= \lambda x.x && \text{(identité)} \\
p_1 &:= \lambda x.\lambda y.x && \text{(première projection)} \\
p_2 &:= \lambda x.\lambda y.y && \text{(deuxième projection)} \\
\circ &:= \lambda f.\lambda g.\lambda x.(f\,(g\,x)) && \text{(composition)} \\
\underline{2} &:= \lambda f.\lambda x.(f\,(f\,x)) && \text{(2ème itération)}
\end{aligned}
$$

Exemples

Quelques termes :

$$
\begin{aligned}
I &:= \lambda x.x && \text{(identité)}\\
p_1 &:= \lambda x.\lambda y.x && \text{(première projection)}\\
p_2 &:= \lambda x.\lambda y.y && \text{(deuxième projection)}\\
\circ &:= \lambda f.\lambda g.\lambda x.(f\,(g\,x)) && \text{(composition)}\\
\underline{2} &:= \lambda f.\lambda x.(f\,(f\,x)) && \text{(2ème itération)}
\end{aligned}
$$

Quelques réductions : en choisissant $f, g, x, y \in \mathcal{V}$ distinctes et $\notin \mathrm{VL}(s,t)$

$$
\begin{aligned}
I\,s &\to_\beta s\\
(p_1\,s)\,t &\to_\beta (\lambda y.s)\,t \to_\beta s\\
(p_2\,s)\,t &\to_\beta I\,t \to_\beta t\\
\lambda h.\circ\,h\,h &\to_\beta \lambda h.(\lambda g.\lambda x.h\,(g\,x))\,h\\
&\to_\beta \lambda h.\lambda x.h\,(h\,x) = \underline{2}
\end{aligned}
$$

Exemples

Quelques termes :

$$
\begin{aligned}
I &:= \lambda x.x && \text{(identité)}\\
p_1 &:= \lambda x.\lambda y.x && \text{(première projection)}\\
p_2 &:= \lambda x.\lambda y.y && \text{(deuxième projection)}\\
\circ &:= \lambda f.\lambda g.\lambda x.(f\,(g\,x)) && \text{(composition)}\\
\underline{2} &:= \lambda f.\lambda x.(f\,(f\,x)) && \text{(2ème itération)}
\end{aligned}
$$

Quelques réductions : en choisissant $f, g, x, y \in \mathcal{V}$ distinctes et $\notin \mathrm{VL}(s,t)$

$$
\begin{aligned}
I\,s &\to_\beta s\\
(p_1\,s)\,t &\to_\beta (\lambda y.s)\,t \to_\beta s\\
(p_2\,s)\,t &\to_\beta I\,t \to_\beta t\\
\lambda h.\circ\,h\,h &\to_\beta \lambda h.(\lambda g.\lambda x.h\,(g\,x))\,h\\
&\to_\beta \lambda h.\lambda x.h\,(h\,x) = \underline{2}
\end{aligned}
$$

- On peut appliquer une fonction à une autre fonction.
- Plusieurs arguments $f(x,y,z) \rightsquigarrow$ plusieurs applications $((f\,x)\,y)\,z$. On écrit : $f\,x\,y\,z$.

Arithmétique

Pour tous termes u et v, on définit l'application itérée de u à v par récurrence :

$$u^0\, v \coloneqq v$$
$$u^{n+1}\, v \coloneqq u\,(u^n\, v).$$

Alors pour tout $n \in \mathbf{N}$, on note $\underline{n} \coloneqq \lambda f.\lambda x.(f^n\, x)$: l'entier de Church numéro n.

On peut alors construire des termes $\underline{\text{add}}$ et $\underline{\text{mul}}$ tels que $\underline{\text{add}}\,\underline{n}\,\underline{p} =_\beta \underline{n+p}$ et $\underline{\text{mul}}\,\underline{n}\,\underline{p} =_\beta \underline{n \times p}$.

Calculabilité

Plus généralement :

Définition

Si f est une fonction partielle de $\mathbf{N}^k$ dans $\mathbf{N}$, on dit que f est représentée par le terme s si, pour tous $(n_1, \ldots, n_k) \in \mathbf{N}$:

- $s\ \underline{n_1} \cdots \underline{n_k} =_\beta \underline{f(n_1, \ldots, n_k)}$ lorsque $f(n_1, \ldots, n_k)$ est défini ;
- la β-reduction ne s'arrête pas sinon.

Théorème

Les fonctions représentables par des λ-termes sont exactement les fonctions calculables par machines de Turing / machines de Minsky / etc.

Types fonctionnels

On considère une grammaire de types fonctionnels simples :

$$A, B, C ::= X \mid A \to B$$

où X varie dans un ensemble de types de base.

λ-termes typables

- Un contexte de typage est une famille finie Γ de types indexée par des variables, qu'on note par exemple $\Gamma = x_1 : A_1, \ldots, x_n : A_n$.
- Un jugement de typage est la donnée $\Gamma \vdash s : A$ d'un contexte Γ, d'un terme s et d'un type A, qu'on lit « dans le contexte Γ, s est typable de type A ».
- Règles de typage :

$$\frac{}{\Gamma, x : A \vdash x : A}\ (var) \qquad \frac{\Gamma, x : A \vdash s : B}{\Gamma \vdash \lambda x.s : A \to B}\ (\lambda) \qquad \frac{\Gamma \vdash s : A \to B \qquad \Gamma \vdash t : A}{\Gamma \vdash s\, t : B}\ (app)$$

Le système T

L'expressivité des termes simplement typés est très limitée.

Le système T de Gödel est obtenu en ajoutant au λ-calcul simplement typé :

- un type de base $\mathbf{N}$;
- des constantes de termes :

$$\begin{aligned} 0 &: \mathbf{N} \\ \mathsf{succ} &: \mathbf{N} \to \mathbf{N} \\ \mathsf{rec}_A &: A \to (\mathbf{N} \to A \to A) \to \mathbf{N} \to A \qquad \text{(pour chaque type } A\text{)} ; \end{aligned}$$

- des règles de réduction :

$$\mathsf{rec}\, a\, f\, 0 \rightsquigarrow a \quad \text{et} \quad \mathsf{rec}\, a\, f\, (\mathsf{succ}\, n) \rightsquigarrow f\, n\, (\mathsf{rec}\, a\, f\, n)$$

Exemples :

$$\begin{aligned} \mathsf{add} &:= \lambda x.\lambda y.(\mathsf{rec}\, y\, \mathsf{succ}\, x) \\ \mathsf{testzero} &:= \lambda x.\lambda y.\lambda z.(\mathsf{rec}\, y\, \lambda ?.z\, x) \end{aligned}$$

Le système T

L'expressivité des termes simplement typés est très limitée.

Le système T de Gödel est obtenu en ajoutant au λ-calcul simplement typé :

- un type de base $\mathbf{N}$;
- des constantes de termes :

$$\begin{aligned} 0 &: \mathbf{N} \\ \mathrm{succ} &: \mathbf{N} \to \mathbf{N} \\ \mathrm{rec}_A &: A \to (\mathbf{N} \to A \to A) \to \mathbf{N} \to A \qquad \text{(pour chaque type } A\text{)} ; \end{aligned}$$

- des règles de réduction :

$$\mathrm{rec}\, a\, f\, 0 \rightsquigarrow a \quad \text{et} \quad \mathrm{rec}\, a\, f\, (\mathrm{succ}\, n) \rightsquigarrow f\, n\, (\mathrm{rec}\, a\, f\, n)$$

Exemples :

$$\begin{aligned} \mathrm{add} &:= \lambda x.\lambda y.(\mathrm{rec}\, y\, \mathrm{succ}\, x) \\ \mathrm{testzero} &:= \lambda x.\lambda y.\lambda z.(\mathrm{rec}\, y\, \lambda ?.z\, x) \end{aligned}$$

Théorème

Tous les termes typés normalisent (la β-réduction s'arrête).

PCF

On peut vouloir conserver un système de types, tout en retrouvant toutes les fonctions calculables.

Il suffit de rajouter au système T un opérateur de point fixe :

- une constante $Y_A : (A \to A) \to A$ pour chaque type A ;
- une règle de réduction $Y_A\, f \rightsquigarrow f\,(Y_A\, f)$.

Ça permet de définir des fonctions par récursion :
pour avoir $f\, x = F[f, x]$, il suffit de poser $f := Y\, \lambda f.\lambda x.F$.

C'est le langage PCF, un prototype de langage de programmation fonctionnelle.

(le pendant simplifié et théorique de la famille ML : SML, OCaml, F#)

Idées clés de la programmation fonctionnelle

- les fonctions sont des valeurs comme les autres
- Curryfication : avoir deux arguments $f(x, y)$ = renvoyer une fonction qui attend le deuxième argument $(f\ x)\ y$
- au lieu d'écrire des boucles bornées (`for`), on itère sur une donnée (récurrence = itération sur les entiers)
- au lieu d'écrire des boucles non bornées (`while`), on utilise la récursion

⇝ En pratique...