Expressivité

Projection : $\overline{\pi_{k,i}} = \lambda x_1.\lambda x_2.\cdots.\lambda x_k.x_i$

Composition : $\overline{f(\ldots, g_j(\ldots x_i \ldots), \ldots)}$ $g_1 \cdots g_m$

Forcer évaluation des paramètres :

$E_m = \lambda x_1.\lambda x_2.\cdots.\lambda x_m.\lambda f.\underbrace{(\cdots(x_1(x_2(\cdots(x_m(\lambda x.x)\cdots)}_{\text{itération identité}} f)x_1)x_2)\cdots)x_m)$

D'où :

$$\lambda x_1.\lambda x_2.\cdots.\lambda x_k.(\cdots((E_m(\overline{g_1}\,\overrightarrow{x}))(\overline{g_2}\,\overrightarrow{x}))\cdots)\overline{f})$$

Exemple : $(\overline{f(g)}\ \overline{0})$ rédution β-normale

Expressivité

Test à 0 : $T_0 = \lambda n.((\overline{n}\ \lambda y.\overline{\pi_{2,2}})\ \overline{\pi_{2,1}})$ on vérifie...

Récursion primitive = ?

On cherche $\overline{f}$ tel que :

$$\overline{f} =_\beta \lambda m.\lambda \overrightarrow{n}.(((T_0\ m)\ (\overline{b}\ \overrightarrow{n}))\ (((\overline{h}\ (pred\ m))\ \overrightarrow{n})\ \underbrace{((\overline{f}\ (pred\ m))\ \overrightarrow{n})}_{\text{appel réc.}})))$$

Expressivité

Théorème (Point fixe)

Il existe Y t. q. pour tout H : $(Y\ H) =_\beta (H\ (Y\ H))$

Exemple : opérateur de Curry $\lambda h.(\lambda x.(h\ (x\ x))\ \lambda x.(h\ (x\ x)))$

Corollaire.

Pour tout H, $(HX) = X$ admet (au moins) une solution

Expressivité

Test à 0 : $T_0 = \lambda x.((\overline{n}\ \lambda y.\overline{\pi_{2,2}})\ \overline{\pi_{2,1}})$ on vérifie...

Récursion primitive = ?

On cherche $\overline{f}$ tel que :

$$\overline{f} =_\beta \lambda m.\lambda \overrightarrow{n}.(((T_0\ m)\ (\overline{b}\ \overrightarrow{n}))\ (((\overline{h}\ (pred\ m))\ \overrightarrow{n})\ \underbrace{((\overline{f}\ (pred\ m))\ \overrightarrow{n})}_{\text{appel réc.}})))$$

$$\overline{f} = (Y\ H_f)$$

Expressivité

Minimisation : ? $\quad f(\overrightarrow{n}) = \min\{m \mid g(m, \overrightarrow{n}) = 0\}$

On cherche A tel que :

$$\begin{array}{lll} ((A\ \overrightarrow{n})\ \overline{m}) & \to^\star_\beta & \overline{m} \qquad\qquad \text{si } g(m, \overrightarrow{n}) = 0 \\ ((A\ \overrightarrow{n})\ \overline{m}) & \to^\star_\beta & ((A\ \overrightarrow{n})\ (succ\ \overline{m})) \qquad \text{sinon} \end{array}$$

Et donc $\overline{f} = \lambda \overrightarrow{x}.((A\ \overrightarrow{x})\ \overline{0})$

Point fixe...

$$H_A = \lambda g.\lambda \overrightarrow{n}.\lambda m.(((T_0\ ((g\ m)\ \overrightarrow{n}))\ m)\ ((g\ (succ\ m))\ \overrightarrow{n}))$$

Expressivité

Théorème

Pour toute fonction récursive partielle $F : \mathbb{N}^k \to \mathbb{N}$,
il existe un λ-terme $\overline{F}$ t. q. :

1. Si F est définie en $(n_1, \ldots, n_k)$ alors
   $(\cdots((\overline{F}\ \overline{n}_1)\ \overline{n}_2)\cdots\overline{n}_k)$ admet $\overline{F(n_1, \ldots, n_k)}$ comme forme normale
2. Si F n'est pas définie en $(n_1, \ldots, n_k)$ alors
   $(\cdots((\overline{F}\ \overline{n}_1)\ \overline{n}_2)\cdots\overline{n}_k)$ n'a pas de forme normale

Atteint grâce à réduction normale

Expressivité

Corollaire.

F λ-représentable $\Leftrightarrow$ F récursive partielle $\Leftrightarrow$ F Turing-calculable

Corollaire.

$=_\beta$ indécidable

λ-calculs typés

- Pur : trop riche
- Avantage : base de langage de programmation typé (`Ocaml`)
- Autre avantage : applications en logique

Dans un premier temps : simplement typé

λ-calcul simplement typé

Ensemble de types de base : B

Ensemble des types construits sur B :

- $t \in B$ alors t type
- t_1 type et t_2 type alors $t_1 \to t_2$ type

Environnement de typage : liste de couples (variable, type)

Jugement de typage : formule logique $\Gamma \vdash E : t$

$[n$: int$]$	$\vdash$	$(\lambda x.x\ n)$: int	valide
$[]$	$\vdash$	$\lambda x.x$: bool	non valide

λ-calcul simplement typé

Validité : axiomes + règles d'inférence

Variable $\dfrac{x : t \in \Gamma}{\Gamma \vdash x : t}$ (première occurrence)

Abstraction $\dfrac{(x : t_1) \cup \Gamma \vdash E : t_2}{\Gamma \vdash \lambda x.E : t_1 \to t_2}$

Application $\dfrac{\Gamma \vdash E_1 : t_1 \to t_2 \quad \Gamma \vdash E_2 : t_1}{\Gamma \vdash (E_1\ E_2) : t_2}$

λ-calcul simplement typé

Validité : axiomes + règles d'inférence

Typable : il existe une dérivation

$[f : int \to bool; n : int] \vdash (f\ n) : bool$ valide

$[] \vdash \lambda x.x : int \to int$

$[] \vdash \lambda x.x : bool \to bool$

$[] \vdash \lambda x.x : (int \to bool) \to (int \to bool)$

$[] \vdash \lambda x.x : t \to t$ pour tout t

En particulier : pas unique $\leadsto$ variables de type

λ-calcul simplement typé

Exo.

- Ω
- $(\lambda x.x\ \lambda x.x)$
- Entiers de Church

Variables de type

Ensemble A t. q. $A \cap B = \emptyset$ variables de type

Application (substitution) de A vers ensemble des types, étendue sur celui-ci :

- $b\sigma = b$ pour $b \in B$
- $(t_1 \to t_2)\sigma = t_1\sigma \to t_2\sigma$

Type principal t pour E si

1. E de type t
2. Pour tout type u de E il existe σ t. q. $t\sigma = u$

Variables de type

Théorème
Le type principal est unique à renommage des variables près

Théorème
La simple typabilité d'un λ-terme est décidable
Si un λ-terme est typable alors il admet un type principal

β-réduction des λ-termes typables

Théorème
Si $E \to_\beta^\star E'$ et $E : t$ alors $E' : t$

Corollaire. (Church-Rosser typé)
Si $E \to_\beta^\star E_1$ et $E \to_\beta^\star E_2$ avec E_1, E_2 bien typés,
alors il existe E_3 bien typé t. q. $E_1 \to_\beta^\star E_3$ et $E_2 \to_\beta^\star E_3$

Dém. par Church-Rosser pur et th.

β-réduction des λ-termes typables

Lemme.

Si $\left\{\begin{array}{lcl} U & : & t \\ x & : & t' \\ V & : & t' \end{array}\right.$ alors $U[x \mapsto V] : t$

Dém. par induction sur U

Normalisation

Théorème (Normalisation faible)
Tout λ-terme typable est faiblement normalisable

Théorème (Normalisation forte)
Tout λ-terme typable est fortement normalisable

Corollaire.
Les fonctions récursives partielles ne sont pas représentables en λ-calcul simplement typé.

Extensions du λ-calcul simplement typé

Manque d'expressivité $\rightsquigarrow$ extensions

- Système T (Gödel, $\simeq$ 40) ;
- Système F (Girard, $\simeq$ 70) ;
- Calcul des constructions ($\simeq$ 80) ;
- ...

Inférence ?

Vérification ?

Système T

Ajout de récursion (syntaxe supp.)

Types : $B = \{int, bool\}$

Termes :

- 0
- $S(t)$ où t λ-terme
- *true* et *false*
- rec(n, t, u) où n, t, u λ-termes
- *if*(b, t, u) où b, t, u λ-termes

Système T

Sémantique : rec opérateur de récursion

Ajout à β de la ι-réduction :

$$
\begin{array}{lcl}
\mathrm{rec}(0, t, u) & \rightarrow_\iota & t \\
\mathrm{rec}(S(n), t, u) & \rightarrow_\iota & ((u\ \mathrm{rec}(n, t, u))\ n) \\
\\
\textit{if}(\textit{true}, t, u) & \rightarrow_\iota & t \\
\textit{if}(\textit{false}, t, u) & \rightarrow_\iota & u
\end{array}
$$

Système T

Nouvelles règles de typage :

$$\frac{}{\Gamma \vdash 0 : int} \qquad \frac{\Gamma \vdash n : int}{\Gamma \vdash S(n) : int}$$

$$\frac{}{\Gamma \vdash \textit{true} : bool} \qquad \frac{}{\Gamma \vdash \textit{false} : bool}$$

$$\frac{\Gamma \vdash n : int \quad \Gamma \vdash t : \tau \quad \Gamma \vdash u : \tau \to (int \to \tau)}{\Gamma \vdash \mathsf{rec}(n, t, u) : \tau}$$

$$\frac{\Gamma \vdash b : bool \quad \Gamma \vdash t : \tau \quad \Gamma \vdash u : \tau}{\Gamma \vdash \textit{if}(b, t, u) : \tau}$$

Système T

Résultats :

- $\to_\beta \cup \to_\iota$ confluente
- $\to_\beta \cup \to_\iota$ fortement normalisante
- Toute fonction récursive totale est représentable

Exemple :

$$\begin{aligned} ADD &= ? \\ MULT &= ? \\ FACT &= ? \end{aligned}$$

Système F

Girard, 1972 Reynolds, 1974

Types : $T : V_T \mid T \to T \mid \forall V_T T$

(Pseudo-) Termes : $\Lambda_T : V \mid (\Lambda_T\ \Lambda_T) \mid (\Lambda_T T) \mid \lambda V : T.\Lambda_T \mid \Lambda V_T.\Lambda_T$

Réduction : $(\lambda x : A.E\ F) \to_\beta E[x \mapsto F]$

Sur les types : $(\Lambda\alpha.E\ A) \to_\beta E[\alpha \mapsto A]$

Vérification : OK

Inférence : NON

Ocaml : F restreint (fragment avec inférence décidable)

Système F

Règles de typage :

Start $\quad \dfrac{x : A \in \Gamma}{\Gamma \vdash x : A}$

App. $\quad \dfrac{\Gamma \vdash E_1 : t_1 \to t_2 \quad \Gamma \vdash E_2 : t_1}{\Gamma \vdash (E_1\ E_2) : t_2}$

Abs. $\quad \dfrac{(x : t_1) \cup \Gamma \vdash E : t_2}{\Gamma \vdash \lambda x : t_1.E : t_1 \to t_2}$

$\forall$-élim. $\quad \dfrac{\Gamma \vdash E : (\forall\alpha.A)}{\Gamma \vdash (E\ B) : A[\alpha \mapsto B]}, B \in \Gamma$

$\forall$-intro. $\quad \dfrac{\Gamma \vdash E : A}{\Gamma \vdash (\Lambda\alpha.E) : (\forall\alpha.A)}, \alpha \notin FV(\Gamma)$

Curry-de Bruijn-Howard

Isomorphisme Curry–de Bruijn–Howard

Lien entre règles de typage et règles de logique intuitionniste

- Abstraction et $(\rightarrow_i)$
- Application et $(\rightarrow_e)$

Extensions du calcul

- Types produits et $(\wedge)$
- Types sommes et $(\vee)$
- ...

Pas de tiers-exclu !

Curry-de Bruijn-Howard

Isomorphisme Curry–de Bruijn–Howard

Propositions $\rightsquigarrow$ types

Preuve d'une proposition $\rightsquigarrow$ habitant du type

Exemple : preuve de $A \Rightarrow B \rightsquigarrow$ fonction de type $A \rightarrow B$

Preuve de $(P \Rightarrow Q) \Rightarrow (Q \Rightarrow R) \Rightarrow (P \Rightarrow R)$:

$$\lambda H_1 \lambda H_2 \lambda p.(H_2\ (H_1\ p))$$

- Vérification : vérification de type
- Possibilité d'extraction de programme certifié

Curry-de Bruijn-Howard

Calcul des constructions inductives (CCI) : λ-calcul typé avec

- Constructeurs de types
- Polymorphisme
- Types dépendants
- Types inductifs

Objets du discours :
Entiers, fonctions, ensembles, propositions, preuves

Discours sur 3 niveaux :
Sortes (`Set`, `Prop`,...), types, termes

Construction des preuves

Langage de tactiques

Commandes de développement de preuve $\rightsquigarrow$ construction d'un λ-terme (terme preuve)

Sécurité : preuve garantie par le typage du terme preuve

Tactique sur un but courant :

- But résolu ou
- Sous-buts