La correspondance de Curry-Howard

(n'est pas épistolaire)

Valentin Maestracci

Hier soir, tard

I2M : Equipe AGLR ! (Sous equipe LDP)

Remontons en arrière

Il y a bien longtemps

Une preuve ensemble

$$\frac{}{\vdash A \implies B \implies A \wedge B}$$

Une preuve ensemble

$$\dfrac{\dfrac{}{A \vdash B \implies A \wedge B}}{\vdash A \implies B \implies A \wedge B} \to_{\mathrm{i}}$$

Une preuve ensemble

$$\dfrac{\dfrac{\dfrac{}{A, B \vdash A \wedge B}}{A \vdash B \implies A \wedge B}\ \to_{\mathrm{i}}}{\vdash A \implies B \implies A \wedge B}\ \to_{\mathrm{i}}$$

Une preuve ensemble

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A, B \vdash A} \quad \dfrac{}{A, B \vdash B}}{A, B \vdash A \wedge B}\ \wedge_{\mathrm{i}}}{A \vdash B \implies A \wedge B}\ \to_{\mathrm{i}}}{\vdash A \implies B \implies A \wedge B}\ \to_{\mathrm{i}}$$

Une preuve ensemble

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A, B \vdash A}\,\mathrm{ax} \qquad \dfrac{}{A, B \vdash B}\,\mathrm{ax}}{A, B \vdash A \wedge B}\,\wedge_{\mathrm{i}}}{A \vdash B \implies A \wedge B}\,\rightarrow_{\mathrm{i}}}{\vdash A \implies B \implies A \wedge B}\,\rightarrow_{\mathrm{i}}$$

Déduction Naturelle

Mais le système minimal, c'est trois règles :

Introduction

$$\dfrac{\dfrac{}{\Gamma, A \vdash B}}{\Gamma \vdash A \implies B} \rightarrow_{\mathrm{i}}$$

Elimination

$$\dfrac{\dfrac{}{\Gamma \vdash A \implies B} \quad \dfrac{}{\Gamma \vdash A}}{\Gamma \vdash B} \rightarrow_{\mathrm{e}}$$

Axiome

$$\dfrac{}{\Gamma, A \vdash A}\ ax$$

Déduction Naturelle

Pour le reste, c'est open bar !

- Opérateur booléens $\vee, \wedge \ldots$
- Quantificateurs $\forall, \exists, \ldots$ (a tout les ordres)
- La fameuse logique linéaire : $!, ?, \otimes, \ldots$
- Broccolis : $\wr\!\!\!\bigcirc$

Le premier langage de programmation !

Introducing : Lambda Calcul ™*

*(Pas vraiment, c'est open source...)

Prenons les fonctions au sérieux

Petite notation

Math	$f : x \longrightarrow x + 2$	$f(3)$	$= 3 + 2 = 5$
Info	$(\lambda x.x + 2)$	$(\lambda x.x + 2)3$	$\rightsquigarrow 3 + 2 \rightsquigarrow 5$

Exemples :

- $\lambda x.x$
- $\lambda x.(\lambda y.y)$
- $(\lambda x.(x)x) = \delta$
- $(\delta)\delta$

Calculons !

$$(\lambda x.x + 3)5$$

$$(\lambda x.\lambda y.y)5$$

$$(\delta)\delta = (\lambda x.(x)x)\delta$$

Formellement

Un programme p peut être :

$$\text{(Var)}\ \ \mathrm{x}$$

$$\text{(Abs)}\ \ \lambda x.q$$

$$\text{(App)}\ \ (q)r$$

Avec la règle

$$(\lambda x.p)q \rightsquigarrow p[x \leftarrow q]$$

Formellement

Un programme p peut être :

$$\text{(Var)}\ \ \mathrm{x}$$

$$\text{(Abs)}\ \ \lambda x.q$$

$$\text{(App)}\ \ (q)r$$

Avec la règle

$$(\lambda x.p)q \rightsquigarrow p[x \leftarrow q]$$

...

On peut vraiment tout faire ?

On peut tout faire !

Tout les programmes*

Par exemple les entiers :

Peano	0	$S0$	$\ldots$
Von Neumann	$\emptyset$	$\{\emptyset\}$	$\ldots$
Church	$\lambda f.\lambda x.x$	$\lambda f.\lambda x.(f)x$	$\ldots$

L'addition :

$$plus := \lambda m.\lambda n.\lambda f.\lambda x.((((((m)f)n)f)x)$$

Etc...

*(de $\mathbb{N}$ dans $\mathbb{N}$)

Génial ! On peut vraiment tout faire !

Tout est permis ! Tout est programmable !

Génial ! On peut vraiment tout faire !

Tout est permis ! Tout est programmable !

Mais du coup on peut programmer n'importe quoi...

Un type pas sympa

$$f : x \longrightarrow x + 2$$

Question :

Un type pas sympa

$$f : x \longrightarrow x + 2$$

Question :

$$f(SO(3))$$

Un type pas sympa

$$f : x \longrightarrow x + 2$$

Question :

$$f(SO(3))$$

$$f(S)$$

Un type pas sympa

$$f : x \longrightarrow x + 2$$

Question :

$$f(SO(3))$$

$$f(S)$$

$$f(PSh(Set))$$

Un type pas sympa

$$f : x \longrightarrow x + 2$$

Question :

$$f(SO(3))$$

$$f(S)$$

$$f(PSh(Set))$$

C'est "mal typé"

Une personne franchement agréable

On va imposer des types a nos termes, et le compilateur vérifiera que tout est bien typé !

Nos nouveaux termes :

$$\text{(Var)} \quad \text{x :A}$$

$$\text{(Abs)} \quad \lambda(x : A).q$$

$$\text{(App)} \quad (q)r$$

A nous de vérifier !

$$\vdash \lambda(x : A).\lambda(y : B).x : A \implies B \implies A$$

A nous de vérifier !

$$\dfrac{\dfrac{}{x : A \vdash \lambda(y : B).x : B \implies A}}{\vdash \lambda(x : A).\lambda(y : B).x : A \implies B \implies A} \implies \text{i}$$

A nous de vérifier !

$$\dfrac{\dfrac{\dfrac{}{x : A, y : B \vdash x : A}\,\text{ax}}{x : A \vdash \lambda(y : B).x : B \implies A}\implies \text{i}}{\vdash \lambda(x : A).\lambda(y : B).x : A \implies B \implies A}\implies \text{i}$$

$$\dfrac{\dfrac{\dfrac{}{\ldots : A, \ldots : B \vdash \ldots : A}\ \text{ax}}{\ldots : A \vdash \ldots : B \implies A}\ \implies \text{i}}{\vdash \ldots : A \implies B \implies A}\ \implies \text{i}$$

Curry-Howard

$$\frac{\overline{\Gamma, A \vdash B}}{\Gamma \vdash A \implies B} \rightarrow_{\mathrm{i}} \qquad \frac{\overline{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \lambda(x : A).t : A \implies B}\ Abs$$

$$\frac{\overline{\Gamma \vdash A \implies B} \quad \overline{\Gamma \vdash A}}{\Gamma \vdash B} \rightarrow_{\mathrm{e}} \qquad \frac{\overline{\Gamma \vdash f : A \implies B} \quad \overline{\Gamma \vdash e : A}}{\Gamma \vdash (f)e : B}\ App$$

$$\frac{}{\Gamma, A \vdash A}\ ax \qquad \frac{}{\Gamma, x : A \vdash x : A}\ Var$$

Le gros résultat

Un programme bien typé est un programme bien élevé !

Preservation du typage

$$\vdash t : A \text{ et } t \rightsquigarrow t' \text{ alors } \vdash t' : A$$

Terminaison

$$\vdash t : A \implies t \in SN$$

Tout va bien

$$\vdash t : \mathbb{N} \implies t \rightsquigarrow^* n$$

Mais c'est trop restrictif !

On élimine aussi les bad boy stylés des programmes...

Mais c'est trop restrictif !

On élimine aussi les bad boy stylés des programmes...

Un exemple :

Mais c'est trop restrictif !

Les types forcent le programme a se comporter d'une certaine manière.

Mais c’est trop restrictif !

Les types forcent le programme a se comporter d’une certaine manière.

Ils sont "prescriptifs"

Mais c’est trop restrictif !

Les types forcent le programme a se comporter d’une certaine manière.

Ils sont "prescriptifs"

Mais en tant que programmeur c’est plus une aide, une indication sur le comportement du programme

Mais c'est trop restrictif !

Les types forcent le programme a se comporter d'une certaine manière.

Ils sont "prescriptifs"

Mais en tant que programmeur c'est plus une aide, une indication sur le comportement du programme

$\Longrightarrow$ On peut regarder des types "descriptifs" ?

Changer de paradigme

La réalisabilité (linéaire chez moi)

- Modèle de calcul +
- Type = Comportement du programme (infinitaire...) +
- Théorème d'adéquation : Si $t \vdash A$ alors $t \Vdash A$

A quoi ça sert tout ça ?

On en sort des trucs !

- Nouveaux type = nouvelle logique !
- Informations sur les programmes !
- Contenu calculatoire des preuves !

D'après vous, quel "contenu calculatoire" le théorème des valeurs intermédiaires à t'il ?