Outils Logiques - L1 MIASHS - L'isomorphisme de Curry-Howard

Romain Péchoux

1 Le lambda-calcul

1.1 Syntaxe

On commence par étudier un langage de programmation, le lambda-calcul, introduit par Church dans les années 30:

$$M, N, \ldots := x \mid (\lambda x.M) \mid (M\ N)$$

où x appartient à un ensemble de variables.

Un programme généré par la grammaire ci-dessus est appelé *lambda-terme* (ou terme pour faire plus court).

Dans la syntaxe ci-dessus le terme $(\lambda x.M)$, est appelé *abstraction* et correspond moralement à la définition d'une fonction mathématique. Par exemple, la fonction identité $f(x) = x$ peut être représenté par le terme $(\lambda x.x)$ (lire "à x on associe x") ou la fonction constante $g(x) = 5$ peut être représenté par le terme $(\lambda x.5)$. Ici, on triche un peu car la constante "5" n'appartient pas à la syntaxe. Nous éclaircirons ce point ultérieurement. A noter, que l'utilisation d'une abstraction évite d'avoir à nommer les objets (les fonctions). Par exemple, nous n'avons pas à donner des noms f, g aux lambda-termes.

Dans la syntaxe ci-dessus, le terme $(M\ N)$, est appelé *application* et correspond moralement à l'application d'une fonction mathématique. Par exemple, $((\lambda x.x)\ 6)$ revient à appliquer la fonction identité à la valeur 6 (donc c'est l'application $f(6)$ en reprenant l'exemple ci-dessus). La valeur de cette application est donc a priori 6.

Lorsqu'une variable apparaît sous un lambda, on dit qu'elle est *capturée*. Par exemple, dans $\lambda x.x$, x est capturée. En revanche, dans dans $\lambda x.y$, y n'est pas capturée. On dit que y est *libre*.

Nous nous intéresserons aux lambda-termes dont toutes les variables sont capturées. On parle de terme *clos*. Un terme clos est moralement un programme dont toutes les variables ont été correctement déclarées. En reprenant l'exemple ci-dessus, $(\lambda x.x)$ est clos. En revanche, $(\lambda x.y)$ ne l'est pas car y est libre.

⚠ Attention, ça se complique : $(\lambda x.(x\ (\lambda x.x))$ est un terme clos mais il est équivalent à $(\lambda x.(x\ (\lambda y.y))$. Chaque occurrence d'une variable est liée par la lambda abstraction la plus proche. Donc, le terme $(x\ (\lambda x.x))$ n'est pas clos car il s'agit de $x\ (\lambda y.y)$.

1.2 Sémantique

La principale règle d'évaluation d'un lambda-terme s'appelle la beta-réduction, notée $\rightarrow_\beta$. Elle énonce qu'on peut réduire l'application d'une abstraction à un terme et se formalise ainsi :

$$((\lambda x.M)\ N) \rightarrow_\beta M\{N/x\}.$$

où la notation $M\{N/x\}$ est le terme obtenu à partir de M en remplaçant toutes les occurrences libres de x par N.

Exemple :

$$((\lambda x.x + x)\ 8) \rightarrow_\beta x + x\{8/x\} = 8 + 8 = 16$$

A noter qu'il existe plusieurs stratégies d'évaluation d'un programme mais ces considérations sortent du cadre de ce cours.

Par ailleurs, le lambda-calcul est *Turing complet* et calcule donc toutes les fonctions calculées par une machine de Turing. Cela signifie qu'il permet de calculer toutes les fonctions pouvant être implémentées sur un langage de programmation standard (C, Java, ...).

2 Le lambda-calcul typé

Dans l'état actuel des choses, les programmes peuvent un peu faire n'importe quoi. Par exemple:

- que signifie $(\lambda x.(x\ x))$?
- que fait le terme $((\lambda x.(x\ x))\ (\lambda x.(x\ x)))$?

On peut restreindre certains de ces comportements non désirables en typant les programmes (d'où l'intérêt d'avoir des types dans les langages comme C, Java, etc).

On suppose, pour l'instant, que l'on dispose des types suivants :

$$F, G, \ldots := \alpha \mid (F \rightarrow G)$$

où α est une variable de type. Pour faire simple, fixons $\alpha = \mathbb{N}$. Par exemple, $\mathbb{N}$, $\mathbb{N} \rightarrow \mathbb{N}$, et $(\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N}))$ sont des types. $\mathbb{N}$ représente les entiers, $\mathbb{N} \rightarrow \mathbb{N}$ représente les fonctions des entiers vers les entiers, $(\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N}))$ représente les fonctions des entiers vers les fonctions des entiers vers les entiers.

Un environnement de typage Γ est une liste de la forme $x_1 : F_1, \ldots, x_n : F_n$ où x_i est une variable et F_i un type. Les variables sont supposées distinctes, $i \neq j$ implique $x_i \neq x_j$.

On peut définir un système de type simple sur le lambda-calcul comme suit:

$$\frac{}{\Gamma, F, \Gamma' \vdash_{\mathrm{NJ}} F}\ (\text{Axiom})$$

$$\frac{\Gamma \vdash M : F \to G \qquad \Gamma \vdash N : F}{\Gamma \vdash (M\ N) : G}\ (\text{App})$$

$$\frac{\Gamma, x : F \vdash M : F}{\Gamma \vdash (\lambda x.M) : F \to G}\ (\text{Abs})$$

Ceci élime les contre-exemples ci-dessus. En effet, l'environnment de typage ne permet pas à une variable d'avoir deux types différents et donc de typer l'application $(x\ x)$.

Exemple de dérivation de typage:

$$\dfrac{\dfrac{\dfrac{}{x : \mathbb{N}, y : \mathbb{N} \vdash x : \mathbb{N}}\ (\text{Var}}{x : \mathbb{N} \vdash (\lambda y.x) : \mathbb{N} \to \mathbb{N}}\ (\text{Abs})}{\vdash (\lambda x.(\lambda y.x)) : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})}\ (\text{Abs})$$

Ce système de type assure une propriété de terminaison sur les programmes. Si un programme type alors il termine (c'est-à-dire qu'on ne peut pas appliquer indéfiniment la règle $\to_\beta$).

3 Correspondance de Curry-Howard

3.1 Présentation générale

La correspondance (l'isomorphisme) entre le lambda-calcul typé et la logique intuitionniste (déduction naturelle) a été énoncée par Curry (pour être précis, ce dernier a montré une correspondance entre les systèmes logiques à la Hilbert et la logique combinatoire) puis prouvée explicitement par Howard en 1969.

Nous regardons cette correspondance règle par règle entre DJ (notation de Prawitz) et système de type.

La règle d'axiome vs la règle de typage d'une variable :

$$\frac{}{\Gamma, F, \Gamma' \vdash_{\text{NJ}} F}\ (\text{Axiom}) \quad \approx \quad \frac{}{\Gamma, x : F, \Gamma' \vdash x : F}\ (\text{Var})$$

L'élimination de l'implication vs la règle de typage d'une application :

$$\frac{\Gamma \vdash_{\text{NJ}} F \Rightarrow G \qquad \Gamma \vdash_{\text{NJ}} F}{\Gamma \vdash_{\text{NJ}} G}\ (\Rightarrow\text{E}) \quad \approx \quad \frac{\Gamma \vdash M : F \to G \qquad \Gamma \vdash N : F}{\Gamma \vdash (M\ N) : G}\ (\text{App})$$

L'introduction de l'implication vs la règle de typage d'une abstraction :

$$\frac{\Gamma, F \vdash_{\text{NJ}} G}{\Gamma \vdash_{\text{NJ}} F \Rightarrow G}\ (\Rightarrow\text{I}) \quad \approx \quad \frac{\Gamma, x : F \vdash M : F}{\Gamma \vdash (\lambda x.M) : F \to G}\ (\text{Abs})$$

Et on poursuit ainsi pour chaque connecteur logique...

L'isomorphisme de Curry-Howard stipule donc qu'à toute preuve d'une formule correspond une dérivation de typage d'un programme, et réciproquement.

3.2 Extensions aux autres connecteurs

Pour la simplicité du cours, on a montré une correspondance avec la version la plus simple du lambda-calcul et un sous-ensemble de DJ. Il est possible d'étendre cette correspondance à tout DJ en augmentant un peu la grammaire de notre langage :

$$\begin{aligned} M, N, L \ldots := {} & x \mid (\lambda x.M) \mid (M\ N) \\ & \mid \langle M, N \rangle \mid \text{fst}\ M \mid \text{snd}\ M \\ & \mid \text{inl}\ M \mid \text{inr}\ M \mid \text{case}\ M\ \text{of}\ (\text{inl}\ u) : N, (\text{inr}\ v) : L \end{aligned}$$

3.2.1 La conjonction

Ici $\langle M, N \rangle$ représente une paire de M et de N tandis que fst et snd représente les projecteurs correspondants.

On peut étendre la sémantique du lambda-calcul par les réductions suivantes:

$$\begin{aligned} \text{fst}\ \langle M, N \rangle &\to M \\ \text{snd}\ \langle M, N \rangle &\to N \end{aligned}$$

Dans ce cas précis, la correspondance de Curry-Howard devient :

L'élimination de la conjonction vs la règle de typage d'un projecteur :

$$\frac{\Gamma \vdash_{\text{NJ}} F \wedge G}{\Gamma \vdash_{\text{NJ}} F}\ (\wedge\text{E}) \quad \approx \quad \frac{\Gamma \vdash M : F \wedge G}{\Gamma \vdash \text{fst}\ M : F}\ (\text{fst})$$

$$\frac{\Gamma \vdash_{\text{NJ}} F \wedge G}{\Gamma \vdash_{\text{NJ}} G}\ (\wedge\text{E}) \quad \approx \quad \frac{\Gamma \vdash M : F \wedge G}{\Gamma \vdash \text{snd}\ M : F}\ (\text{snd})$$

L'introduction de la conjonction vs la règle de typage d'une paire :

$$\frac{\Gamma \vdash_{\text{NJ}} F \qquad \Gamma \vdash_{\text{NJ}} G}{\Gamma \vdash_{\text{NJ}} F \wedge G}\ (\wedge\text{I}) \quad \approx \quad \frac{\Gamma \vdash M : F \qquad \Gamma \vdash N : G}{\Gamma \vdash \langle M, N \rangle : F \wedge G}\ (\text{pair})$$

A noter que les paires et projecteurs peuvent être codés dans le lambda-calcul initial mais cela est un peu plus complexe.

3.2.2 La disjonction

Ici case M of $(\text{inl}\ u) : N, (\text{inr}\ v) : L$ représente un choix entre la possibilité d'évaluer M en $(\text{inl}\ u)$ ou en $(\text{inr}\ v)$ tandis que inl et inr sont des "injections" permettant d'encapsuler un terme dans ce choix binaire.

On peut étendre la sémantique du lambda-calcul par les réductions suivantes:

$$\text{case}\ M\ \text{of}\ (\text{inl}\ u) \to N, (\text{inr}\ v) \to L \to_{case} \begin{cases} N\{N'/u\} & \text{if } M \to (\text{inl}\ N') \\ L\{L'/v\} & \text{if } M \to (\text{inr}\ L') \end{cases}$$

Dans ce cas précis, la correspondance de Curry-Howard devient :
L'introduction de la disjonction vs la règle de typage d'une injection :

$$\frac{\Gamma \vdash_{\mathrm{NJ}} F}{\Gamma \vdash_{\mathrm{NJ}} F \vee G}\ (\vee\mathrm{I}) \quad \approx \quad \frac{\Gamma \vdash M : F}{\Gamma \vdash \mathrm{inl}\ M : F \vee G}\ (\mathrm{inl})$$

$$\frac{\Gamma \vdash_{\mathrm{NJ}} G}{\Gamma \vdash_{\mathrm{NJ}} F \vee G}\ (\vee\mathrm{I}) \quad \approx \quad \frac{\Gamma \vdash M : G}{\Gamma \vdash \mathrm{inr}\ M : F \vee G}\ (\mathrm{inr})$$

L'élimination de la disjonction vs la règle de typage d'un case :

$$\frac{\Gamma \vdash_{\mathrm{NJ}} F \vee G \qquad \Gamma, F \vdash_{\mathrm{NJ}} H \qquad \Gamma, F \vdash_{\mathrm{NJ}} H}{\Gamma \vdash_{\mathrm{NJ}} H}\ (\vee\mathrm{E})$$

$$\approx$$

$$\frac{\Gamma \vdash M : F \vee G \qquad \Gamma, u : F \vdash N : H \qquad \Gamma, v : F \vdash L : H}{\Gamma \vdash \mathrm{case}\ M\ \mathrm{of}\ (\mathrm{inl}\ u) \to N, (\mathrm{inr}\ v) \to L : H}\ (\mathrm{case})$$

3.2.3 Validité et contradiction

Pour trouver une correspondance avec la validité et la contradiction, il nous faut encore étendre la grammaire :

$$M, N, \ldots := \ldots \mid \langle\rangle \mid \mathrm{abort}\ M$$

mais aussi les types:

$$F, G, \ldots := \alpha \mid (F \to G) \mid \top \mid \bot$$

A noter que ces deux constructeurs n'ont pas vraiment de contenu calculatoire. $\langle\rangle$ représente le vide et abort M arrête le calcul suite à une erreur (qu'on appelle exception).

Introduction de la validité vs typage du vide (void):

$$\frac{}{\Gamma \vdash_{\mathrm{NJ}} \top}\ (\top\mathrm{I}) \quad \approx \quad \frac{}{\Gamma \vdash \langle\rangle : \top}\ (\mathrm{Void})$$

Elimination de la contradiction vs typage du abort (exception) :

$$\frac{\Gamma \vdash_{\mathrm{NJ}} \bot}{\Gamma \vdash_{\mathrm{NJ}} F}\ (\neg\mathrm{E}) \quad \approx \quad \frac{\Gamma \vdash M : \bot}{\Gamma \vdash \mathrm{abort}\ M : F}\ (\mathrm{Abort})$$

3.2.4 Négation

Pour la négation, c'est un peu plus subtile car il est difficile de trouver un constructeur correspondant. En réalité, on part du fait que $\neg A$ peut être codée par $A \Rightarrow \bot$ et que la logique intuitionniste peut donc être considérée sans la négation.