Raisonnons par induction

1 Rappel

Lors de l'épisode précédent, nous avions vu :
-le schéma général d'une définition inductive :

$$objet ::= brique_1 \mid brique_2 \mid \ldots \mid Constructeur_1(objet_1, objet_2, \ldots) \mid Constructeur_2(objet_1, objet_2) \mid \ldots$$

-la syntaxe du λ-calcul :

$$T, U ::= x \mid \lambda x.(T) \mid TU$$

-la β-réduction :

$$(\lambda x.T)u \longrightarrow_\beta T\{x := U\}$$

Nous allons ici essayer d'approcher la notion de preuve par induction structurelle, sans trop rentrer dans les détails techniques, puis nous verrons comment construire certains objets un peu particulier par induction.

2 Preuve par induction

Nous avons maintenant construits de beaux objets, et en sommes très content, mais comment peut-on prouver des résultats sur lesdits objets ? "Par induction", répond-on mécaniquement lorsque l'on est rompu à ce type de formalisme. Pour ceux qui connaissent la récurrence, l'idée est la même. Pour les autres, une fois que vous aurez compris ça, vous aurez compris la récurrence, sans le savoir !

L'idée fondamentale sur laquelle tout repose est que si l'on nous donne un objet, celui-ci a été construit avec des briques et des constructeurs. Et donc que l'on va pouvoir le déconstruire. Il suffit donc, pour prouver un résultat sur tous les objets de ce type, de montrer le résultat pour les briques de base (étape dite d'*initialisation*), et de montrer que les constructeurs le préserve (moteur de l'induction).

Prenons un exemple tout simple, soit la propriété suivante $P(n) \equiv$"n *ou* $n+1$ *est pair*". Comment prouve-t'on ceci ? On repart de la définition inductive des entiers : $n ::= 0 \mid S(n)$.

- $P(0)$ est vraie, car 0 est pair
- On veut montrer $P(n)$ pour $n = S(n')$. On peut supposer ("par induction", dit-on en langage soutenu), n' étant structurellement plus petit que n, avoir déjà montré $P(n')$. Donc n' ou $n'+1$ est pair. Si $n'+1 = n$ est paire, on a $P(n)$. Si n' est pair, $n' = 2p$, alors $n+1 = n'+2 = 2p+2 = 2(p+1)$ est pair. On a donc $P(n)$. Dans tous les cas, on a donc $P(n)$
- On en déduit donc que P est vraie pour tous les n

Je n'ai pas en tête d'induction simple sur les λ-termes qui puisse servir d'exemple, si c'est un jour le cas, je le rajouterai.

3 Construction inductive

3.1 Substitution

Pour définir la β-réduction, on se sert d'une notion relativement naturelle de substitution $\{x := U\}$. J'avais aussi mentionné dans les notes qu'on pouvait définir formellement la substitution par induction. L'idée d'une telle construction est exactement la même que pour effectuer une preuve : on définit ce qu'est la substitution pour les cas de bases (ici des variables), et on explique comment substituer dans un termes complexes à base de constructeurs en se servant de la substitution dans les sous-termes. Ce qui donne :

$$\begin{array}{rcl|rcl} y\{x := U\} & = & y & (\lambda y.T)\{x := U\} & = & \lambda y.(T\{x := U\}) \\ x\{x := U\} & = & U & (T_1T_2)\{x := U\} & = & (T_1\{x := U\})(T_2\{x := U\}) \end{array}$$

On voit bien que pour définir la substitution sur un "gros terme", on se sert de celle définie sur des sous-termes, donc plus petits. On peut par exemple de cette manière programmer très facilement la substitution.

Exercice 1. *Vous pouvez essayer de définir de même ce qu'est une forme normale de façon formelle. Cependant, vous allez peut-être vous rendre compte (dans le cas de l'application) qu'il y a un léger problème caché. Mais à cela rien d'anormal, puisqu'il faut se rappeler que certains termes (e.g.* Ω*) n'ont pas de forme normale.*

3.2 Typage

Souvenez-vous, afin de mieux comprendre le lambda-calcul, on peut faire le parallèle suivant avec les fonctions qui vous sont familières :

$$\begin{array}{ccccc} f & : & x & \mapsto & f(x) \\ \wr & & \wr & & \wr \\ \lambda x.t & : & U & \mapsto & t\{x := U\} \end{array}$$

Or, souvenez-vous aussi qu'usuellement, on aime bien savoir d'où partent et où arrivent les fonctions (dans $\mathbb{N}$, dans $\mathbb{R}$, dans le plan ($\mathbb{R}^2$, etc...). Et bien en λ-calcul, c'est pareil, et ça s'appelle le typage. Moralement, l'idée est de décrire quelle sorte d'argument on attend et quelle sorte de résultat on va rendre.
La notation $T : A$ signifie T *est de type* A. Un terme de type $A \to B$ sera tout naturellement une fonction de A dans B, etc. Et bien évidemment, on ne peut construire le typage que par induction ! Pour ce faire, on va utiliser ce qu'on appelle un *arbre d'inférence* (et dont on s'en resservira pour faire des preuves formelles plus tard). Une *règle d'inférence* est de la forme :

$$\frac{Hypothèses_1 \vdash Résultat_1 \qquad Hypothèses_2 \vdash Résultat_2 \qquad ...}{Hypothèses \vdash Résultat}\ \texttt{Nom de la règle}$$

Ce qui se lit : si, sous les hypothèses1, j'ai prouvé le résultat1, sous les hypothèses2 le résultat2, etc... alors je peux en déduire sous les hypothèses finales le résultat final.
Dans notre cas précis, les hypothèses (que nous noterons Γ) correspondront à une liste d'information sur des variables ($\Gamma \equiv (x_1 : A_1), (x_2 : A_2), lodts, (x_n : A_n)$). Les règles sont les suivantes :

$$\frac{}{\Gamma \vdash x : A}\ (x : A) \in \Gamma \qquad \frac{\Gamma \vdash T : A \to B \qquad \Gamma \vdash U : A}{\Gamma \vdash TU : B}\ @ \qquad \frac{\Gamma, x : A \vdash T : B}{\Gamma \vdash \lambda x.T : A \to B}\ \lambda$$

Pour être sur de bien comprendre ce qu'elles font, un petit exemple, le typage de 1, qui attend son successeur, un zéro, et renvoie un entier naturel (nat) :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{z : nat, s : nat \to nat \vdash s : nat \to nat} \qquad \dfrac{}{z : nat, s : nat \to nat \vdash z : nat}\ @}{z : nat, s : nat \to nat \vdash s(z) : nat}}{s : nat \to nat \vdash \lambda z.s(z) : nat \to nat}\ \lambda}{\vdash \lambda sz.s(z) : (nat \to nat) \to nat \to \mathbb{N}}\ \lambda$$

Pour construire un tel arbre, en fait, on part d'en bas, on remonte sans noter les types en appliquant mécaniquement les règles (on n'a jamais le choix) jusqu'à atteindre les variables tout en haut. Là on décide du type que l'on donne aux variables, selon ce que l'on attend d'elle (par exemple ici le zéro est forcément un naturel, et le successeur une fonction qui prendre un naturel et en renvoie un autre).

Exercice 2. *Si vous voulez être sur que vous avez compris, vous pouvez prendre des termes simples (swap, T) ou un peu moins simples (l'addition, la multiplication) et essayer de donner leur typage, puis de le prouver avec un arbre d'inférence. Que se passe-t'il avec Ω ?*

À titre culturel

En fait, tout n'est pas typable en λ-calcul. C'est notamment le cas d'Ω (on peut le prouver, en supposant qu'il y en ait un, puis en raisonnant sur ce qu'il devrait être, on abouti à une contradiction). Cependant, on gagne beaucoup de propriétés intéressantes sur les λ-termes. Par exemple, on sait montrer que tout terme typable admet forcément une forme normale (la preuve repose sur le fait qu'à chaque pas de β-réduction, la somme de la taille des types des objets diminue strictement).

4 Le mot de la fin

C'est tout ce que je comptais vous raconter sur le λ-calcul et l'induction, qui sont des outils assez prisés des logiciens dans la recherche. Si j'en ai le temps, je vous montrerai peut-être en marge de la séance sur les preuves formelles comment on peut, avec le λ-calcul typé, réaliser une correspondance entre les preuves et les programmes (appellée l'isomorphisme de Curry-Howard).

Next episode : *Preuves Formelles.*