Fondements de la programmation

Exercices 9 et 10 lambda-calcul

Stratégie d'évaluation : **appel par valeur.** Nous allons maintenant restreindre la β-réduction de façon à ce qu'elle corresponde à la façon dont les programmes sont évalués dans la plupart des langages de programmation. On distingue un ensemble particulier de termes appelés valeurs au delà desquels on ne (β-)réduit plus. En l'absence d'extensions du lambda-calcul, qui viendront après, les valeurs sont tous les termes qui correspondent à une abstraction :

$$v := \lambda x.t$$

Intuitivement, on ne réduit pas dans le corps des fonctions tant qu'elles ne sont pas complètement appliquées.

On se donne ensuite des règles d'évaluation qui déterminent exactement l'ordre dans lequel il faut faire les réductions dans un terme.

$$\frac{}{(\lambda x.t_1\ v_2) \to t_1[x := v_2]}$$

$$\frac{t_1 \to t_1'}{(t_1\ t_2) \to (t_1'\ t_2)} \qquad \frac{t_2 \to t_2'}{(v_1\ t_2) \to (v_1\ t_2')}$$

Le point important dans cet ensemble de règles est qu'il n'y a pas de règle qui permette de réduire sous une abstraction.

Nous reviendrons plus en détails sur les différentes stratégies d'évaluation possibles.

Unicité du typage. Nous utilisons désormais une version du lambda-calcul simplement typé où chaque variable liée par un lambda est annotée par un type, comme ceci : $\lambda x[T].t$, de telle sorte que T soit le type associé à la variable x au moment du jugement de typage :

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x[A].t : A \Rightarrow B}\ \text{abs.}$$

De cette façon, étant donné un contexte de typage Γ, et un terme t ainsi annoté il existe au plus un type T tel que $\Gamma \vdash t : T$.

Nous allons maintenant donner des **extensions du lambda-calcul simplement typé** de façon à en faire un langage de programmation, Turing-complet et ayant les caractéristiques d'un langage de type ML (les langages ML comme SML, CAML sont inspirés des extensions du lambda-calcul et en particulier du langage PCF, *programming computable functions*).

Unit. On ajoute une valeur `unit` de type, `Unit`.

$$t := \ldots \mid \texttt{unit}$$
$$v := \ldots \mid \texttt{unit}$$
$$T := \ldots \mid \texttt{Unit}$$

$$\frac{}{\Gamma \vdash \texttt{unit} : \texttt{Unit}}$$

Ceci nous permet d'introduire une *forme dérivée* :

$$t_1; t_2 \stackrel{\text{def}}{=} (\lambda x[\texttt{Unit}].t_2\ \ t_1) \quad (x \notin FV(t_2)).$$

Cette mise en séquence, $t_1; t_2$ a pour effet d'évaluer t_1, de jeter son résultat, puis d'évaluer t_2. Les règles d'évaluation et de typage induites sur la forme sont :

$$\frac{t_1 \to t_1'}{t_1; t_2 \to t_1'; t_2} \qquad \frac{}{\texttt{unit}; t_2 \to t_2}$$

$$\frac{\Gamma \vdash t_1 : \texttt{Unit} \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1; t_2 : T_2}$$

Let in. On peut également se donner la forme syntaxique `let in`

$$t := \ldots \mid \texttt{let}\ x = t\ \texttt{in}\ t$$

$$\frac{}{\texttt{let}\ x = v_1\ \texttt{in}\ t_2 \to t_2[x := v_1]}$$

$$\frac{t_1 \to t_1'}{\texttt{let}\ x = t_1\ \texttt{in}\ t_2 \to \texttt{let}\ x = t_1'\ \texttt{in}\ t_2}$$

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \texttt{let}\ x = t_1\ \texttt{in}\ t_2 : T_2}$$

Au premier abord, cette forme $\texttt{let}\ x = t_1\ \texttt{in}\ t_2$ est juste du sucre syntaxique pour $(\lambda x[T_1].t_2\ t_1)$. Pour *désugariser* il faudra toutefois trouver l'annotation de type pour λx, donc faire l'inférence de type avant.

Combinateur de point fixe. Pour chaque type T_1 on se donne un opérateur sur les termes

`fix` tel que :

$$t := \ldots \mid (\texttt{fix}\ t)$$

$$\frac{}{(\texttt{fix}\ \lambda x[T_1].t_2) \to t_2[x := (\texttt{fix}\ \lambda x[T_1].t_2)]}$$

$$\frac{t_1 \to t_1'}{\texttt{fix}\ t_1 \to \texttt{fix}\ t_1'}$$

$$\frac{\Gamma \vdash t_1 : T_1 \to T_1}{\Gamma \vdash (\texttt{fix}\ t_1) : T_1}$$

Forme dérivée :

$$\texttt{letrec}\ x = t_1\ \texttt{in}\ t_2 \stackrel{\text{def}}{=} \texttt{let}\ x = (\texttt{fix}\ \lambda x[T_1].t_1)\ \texttt{in}\ t_2$$

On peut construire des types de données composées, le plus simple d'entre eux étant les **paires** :

$$t := \ldots \mid \{t, t\} \mid t.1 \mid t.2$$
$$v := \ldots \mid \{t, t\}$$
$$T := \ldots \mid T_1 \times T_2$$

Règles de réduction des paires :

$$i = 1, 2\ \frac{t \to t'}{t.i \to t'.i} \qquad \frac{}{\{v_1, v_2\}.i \to v_i}\ i = 1, 2$$

$$\frac{t_1 \to t_1'}{\{t_1, t_2\} \to \{t_1', t_2\}} \qquad \frac{t_2 \to t_2'}{\{v_1, t_2\} \to \{v_1, t_2'\}}$$

Règles de typage des paires :

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \{t_1, t_2\} : T_1 \times T_2} \qquad \frac{\Gamma \vdash t : T_1 \times T_2}{\Gamma \vdash t.i : T_i}$$

1 Exercices

Question A. Typage. Typer les termes suivants en utilisant les extensions appropriées du lambda-calcul.

1. $\texttt{let}\ f = \lambda xy.x\ \texttt{in}\ \lambda abc.(f\ (f\ a\ b)\ c)$.
2. $(\lambda xy.x\ \texttt{unit}\ (\lambda z.z)); \lambda xy.y$
3. $(\texttt{fix}\ \lambda x.x)$
4. $(\texttt{fix}\ \lambda xy.x)$
5. $\texttt{letrec}\ f = \lambda ghx.(g\ x\ (f\ g\ h\ (h\ x))\ (h\ x))\ \texttt{in}\ (f\ (\lambda xyz.z)\ (\lambda z.z))$
6. $\lambda x.x.1$
7. $\lambda xy.\{x, y\}$

1.1 Letrec

Question B. Question de partiel. Dans un `letrec` qu'est-ce qui s'apparente à du sucre syntaxique (ou une forme dérivée) et qu'est ce qui est une véritable extension du lambda-calcul simplement typé ?

Question C. Fonctions récursives. Définir factorielle à l'aide d'un `letrec` puis supprimer le sucre syntaxique. De même pour la fonction de Fibonacci.

Question D. Récursion. La plupart du temps un `letrec` sert à définir une fonction, c'est à dire un terme de type $T_1 \Rightarrow T_2$, pour un certain T_1 et un certain T_2. Pouvez-vous trouver une expression définie récursivement et qui soit d'un type autre qu'une flèche ?

1.2 Preuves par induction sur les jugements de typage

Question E. Affaiblissement (importation dans d'autres contextes). Prouver que, en lambda-calcul simplement typé, si $\Gamma \vdash t : T$ alors pour n'importe quel $x \notin FV(t)$, $\Gamma, x : T' \vdash t : T$.

En déduire que si $\Gamma \vdash t : T$ alors $\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash t : T$, pour n'importe quels $x_1, \ldots, x_n \notin FV(t)$.

Question F. Substitution (modularité et liaison.) Prouver que si $\Gamma, x : T' \vdash t : T$ et $\Gamma \vdash t' : T'$, alors $\Gamma \vdash t[x := t'] : T$.