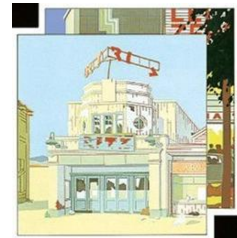
Programmation 1 – TP n° 8
Machine Abstraite

1 De Bruijn



(Refrain) L'α-conversion pose des problèmes d'implémentation parce qu'il est difficile de travailler sur des classes d'équivalence (il est difficile de choisir des représentants qui respectent les règles de réduction).

La représentation en indice de De Bruijn, où une variable est un entier indiquant le nombre de λ entre elle et son lieur, est une solution plus *computational-friendly*.

Q 1.1 Rappelez la syntaxe du λ-calcul en indice de De Bruijn, la substitution et une sémantique à petit pas en appel par valeur.

L'inconvénient principal de représentation en indice de De Bruijn concerne les λ-termes ouverts (ceux contenant des variables libres) ; on va chercher à caractériser les λ-termes clos par un prédicat $\mathcal{C}$ tel que : $\mathcal{C}(M)$ est vrai $\iff$ M est clos.

Q 1.2 Parmi les nombreuses définitions possibles de $\mathcal{C}$, trouvez en une qui soit inductive sur la syntaxe des λ-termes. Indication : il sera nécessaire de définir un prédicat intermédiaire plus général.

2 Sémantique naturelle

Dans la suite, nous aurons besoin d'utiliser une sémantique opérationnelle à grand pas appelée sémantique naturelle.

Q 2.1 Définissez la relation de réduction $_ \Downarrow _$ correspondant à cette sémantique.

Lors de l'implémentation, la substitution textuelle, qui parcourt en entier un terme pour substituer une variable par une valeur, se révèle souvent inefficace. Une meilleure solution consiste à garder la trace des substitutions dans un environnement, on écrire alors $e \vdash M \Downarrow V$ pour dire que dans l'environnement e, le terme M se réduit en la valeur V.

Q 2.2 Comment modifier les valeurs du calcul afin de garder une liaison statique (lexicale) lors de l'exécution (penser au TP n° 4) ?

Q 2.3 Définir une sémantique à grand pas plus proche de l'implémentation.

Q 2.4 Montrez que si $\emptyset \vdash M \Downarrow C$ alors $M \to_v^* \mathcal{E}(C)$ où $\mathcal{E}$ est une traduction à définir.

Pour l'autre sens, on passe par quelques lemmes intermédiaires :

Q 2.5 Si $\mathcal{C}_{|e|}(M)$ et $e \vdash M[V/|e|] \Downarrow C$, alors $V \cdot e \vdash M \Downarrow C$.

Q 2.6 Si $M \to_v M'$ et $\emptyset \vdash M' \Downarrow C$, alors $\emptyset \vdash M \Downarrow C'$ avec $\mathcal{E}(C) = \mathcal{E}(C')$.

Q 2.7 Et enfin, si $M \to_v^* V$ avec V une valeur, alors $\emptyset \vdash M \Downarrow C$ et $\mathcal{E}(C) = V$.

3 Reste sur la scène, comme une machine abstraite

Dans cette section, on définit une machine abstraite relativement bas-niveau dans laquelle on espère compiler le λ-calcul. La machine SECD possède trois composantes : un pointeur de code (une liste d'instructions), un environnement (une liste de valeurs) et une pile (une liste de valeurs).

Un état de la machine est noté $\langle\ c \mid e \mid s\ \rangle$, la liste vide ϵ et l'opérateur "cons" $\cdot$. Le jeu d'instructions est :

$$ACCESS(n) \mid CLOSURE(c) \mid APPLY \mid RETURN$$

où n est un entier et c une liste d'instructions. Les transitions de la machine sont définies par :

$$\begin{array}{rcl}
\langle\ ACCESS(i) \cdot c \mid e \mid s\ \rangle & \longrightarrow_M & \langle\ c \mid e \mid e(i) \cdot s\ \rangle \\
\langle\ CLOSURE(c) \cdot c' \mid e \mid s\ \rangle & \longrightarrow_M & \langle\ c' \mid e \mid (c,e) \cdot s\ \rangle \\
\langle\ APPLY \cdot c' \mid e' \mid v \cdot (c,e) \cdot s\ \rangle & \longrightarrow_M & \langle\ c \mid v \cdot e \mid c' \cdot e' \cdot s\ \rangle \\
\langle\ RETURN \cdot c' \mid e \mid v \cdot c \cdot e' \cdot s\ \rangle & \longrightarrow_M & \langle\ c \mid e' \mid v \cdot s\ \rangle
\end{array}$$

Q 3.1 Définir un schéma de compilation $\mathcal{S}$ des λ-termes en indices de De Bruijn dans le langage SECD.

On cherche maintenant à montrer que cette machine abstraite respecte bien les règles de réduction du λ-calcul en appel par valeur. En réalité, on ne prouvera ici qu'une forme faible de correction :

$$\text{si } M \to_v^* V \text{, alors } \langle\ \mathcal{S}(M) \cdot \epsilon \mid \epsilon \mid \epsilon\ \rangle \longrightarrow_M^* \langle\ \epsilon \mid \epsilon \mid \mathcal{S}'(V) \cdot \epsilon\ \rangle.$$

Q 3.2 Proposez des formes plus fortes de correction.

Q 3.3 Prouvez un théorème plus général que la correction faible :

$$\text{si } e \vdash M \Downarrow V \text{, alors } \langle\ \mathcal{S}(M) \cdot c \mid \mathcal{S}(e) \mid s\ \rangle \longrightarrow_M^+ \langle\ c \mid \mathcal{S}(e) \mid \mathcal{C}(V) \cdot s\ \rangle.$$

3.1 Ajout de fonctionnalités

On souhaite rajouter à la syntaxe du λ-calcul quelques primitives pour en faire un vrai langage de programmation, par exemple :
- des entiers et une opération d'addition,
- un "let .. in",
- etc.

Q 3.4 `GOTO` 1.

3.2 Appel terminal

On considère le λ-terme $(\lambda fx.fx)(\lambda x.x+1)2$.

Q 3.5 A quoi ressemble sa compilation et son exécution ?

Q 3.6 Quel problème se pose ?

Q 3.7 Proposez une optimisation.