UFR 919 Sciences de l'ingénieur
Master Informatique – Spécialité STL
Analyse de Programmes et Sémantique (APS)

TD6 — Les points fixes

Jacques Malenfant, Olena Rogovchenko

1 Récursivité et points fixes

Rappel : L'opérateur de point fixe est défini par $\mathbf{fix} \equiv \lambda f.((\lambda x.(f\ (x\ x)))\ (\lambda x.(f\ (x\ x))))$

Soient les fonctions `f` et `fac` suivantes :

```
f : EV → EV
f(m, n) = if m = n then n else 1 + f(m-1, n)
fac : EV → EV
fac(n) = if n = 0 then 1 else n * fac(n - 1)
```

Pour chacune de ces fonctions :

Q1. Définissez les fonctionnnelles $F_{\texttt{f}}$ et $F_{\texttt{fac}}$ correspondantes.

Q2. Calculez ((**fix** $F_{\texttt{f}}$) 3 1) et ((**fix** $F_{\texttt{fac}}$) 2).

Q3. Donnez pour chacune des fonctionnelles le résultat de l'application de F^0, F^1, F^2 et F^3 à la fonction initiale qui retourne $\perp_{\mathbf{EV}}$ pour toute entrée, et déduisez-en une forme générale pour F^i.

2 Sémantique par point fixe du `while`

Dans le mini-langage impératif des cours 6 et 7, la sémantique de l'instruction `while` est définie comme :

$$execute[\![\texttt{while}\ be\ i]\!]\rho\sigma = (loop\ \sigma)$$
$$\textbf{where}\ loop\ = (\textbf{fix}\ \lambda f.\lambda\sigma.\textbf{if}\ beval[\![be]\!]\rho\sigma)\ \textbf{then}\ (f\ execute[\![i]\!]\rho\sigma)\ \textbf{else}\ \sigma)$$

Q1. En prenant la définition de l'opérateur de point fixe de la question précédente, déroulez l'exécution pour le cas suivant :

$$execute[\![\texttt{while}\ (\texttt{x} < \texttt{1})\ (\texttt{x} := \texttt{x} + \texttt{1})]\!]\rho\sigma$$

avec $\rho_0 = \{x \mapsto 10\}$ et $\sigma_0 = \{10 \mapsto 0\}$.

Q2. En partant de la mémoire initiale $\lambda a.\perp_{\mathbf{SV}}$, obtenez les formes de $loop_0$, $loop_1$, $loop_2$ et $loop_3$. Déduisez-en la forme générale de $loop_n$.

3 Extension du mini-langage

Ajoutez la boucle **for** au mini-langage, avec la même syntaxe que celle qui a été proposée au TD3 :

`for` *id* `:=` *expr* `to` *expr* `step` *expr* `do` *inst*

Définissez la sémantique de la boucle **for** en utilisant l'opérateur du point fixe.

4 Partie TME : le mini-langage

Dans ce TME, nous étudions le mini-langage impératif défini en cours (séance 6). Une implantation en Scheme de la sémantique dénotationnelle de ce mini-langage est disponible dans le fichier `mini-langage.scm` sur le site de l'UE.

4.1 Premiers pas avec l'implantation du mini-langage

Étudiez l'implantation de la sémantique du mini-langage en mettant en correspondance la sémantique vue en cours (transparents) et celle que vous trouverez dans le fichier `mini-langage.scm` disponible sur le site de l'UE. Cette implantation commence par la mise en œuvre de la syntaxe abstraite pour créer les noeuds de l'arbre de syntaxe abstraite. Ensuite, elle définit les domaines sémantiques et leurs fonctions de manipulation. Enfin, il y a les équations sémantiques.

Pour les équations sémantiques, notez que Scheme n'offre pas le filtrage des paramètres syntaxiques que nous utilisons dans les équations formelles pour distinguer les différents cas d'expressions et d'instructions, par exemple. Chaque fonction sémantique est donc implantée par une fonction d'aiguillage qui teste les différentes formes syntaxiques, puis des fonctiosn différentes sont utilisées pour représenter chacune des équations de la sémantique.

Construction de termes

Pour utiliser cette implantation, il faut lui fournir des programmes à exécuter. Nous n'avons pas défini de syntaxe concrète pour ce mini-langage, donc construire un programme se fait par la construction manuelle de son arbre de syntaxe abstraite. Par exemple, pour construire une expression comme 32-10, il faut faire la séquence suivante :

```
(define dix
  (make-compose (make-simple (make-chiffre 'un)) (make-chiffre 'zero)))
(define exp-cons10 (make-constante dix))
(define trente-deux
  (make-compose (make-simple (make-chiffre 'trois)) (make-chiffre 'deux)))
(define exp-cons32 (make-constante trente-deux))
(define exp32moins10 (make-moins exp-cons32 exp-cons10))
(begin (display "exp32moins10 : ")
       (display (((eval exp32moins10) emptyEnv) emptyStore)) (newline))
```

Q1. En vous inspirant de l'exemple précédent, créez en Scheme les arbres de syntaxe abstraite pour les expressions suivantes :

1. la constante 5
2. $10 + 7$
3. $10 < 8$
4. $(10 < 8) \mid (10 = 8)$

Comme on n'a pas de déclaration de variables dans le langage, il faut initialiser manuellement l'environnement avec toutes les variables présentes dans les expressions et initialiser la mémoire aux adresses attribuées aux variables. La séquence suivante alloue une adresse, range la valeur 10 à cette adresse, et lie la variable `x` à l'adresse allouée dans l'environnement :

```
(define a1 0)
(define sigma1 (((updateStore emptyStore) a1) 10))
(define id-x (make-identificateur 'x))
(define rho1 (((extendEnv emptyEnv) id-x) a1))
```

Q2. Considérez la séquence suivante :

```
(define tant-que-vrai
  (make-tant-que
    (make-plus-petit exp-cons0 id-x)
    (make-affectation
      id-x
      (make-moins id-x (make-constante (make-simple (make-chiffre
      'un)))))))

(begin (display "tant-que-vrai : ")
       (let ((sigma (((execute tant-que-vrai) rho1) sigma1)))
         (display (((eval id-x) rho1) sigma)) (newline)))
```

Quel est le terme construit dans la variable **tant-que-vrai** ?

Q3. En vous inspirant de ce terme, créez en Scheme les arbres de syntaxe pour les instructions suivantes. Évaluez ensuite ces arbres en initialisant correctement la mémoire et l'environnement :

1. a := 1 ; b := 2 ; c := a+b ;
2. if x > y then c = x else c = y
3. while (x!= 0) do x := x-1 ;

Factorielle bis

Construisez l'arbre de syntaxe abstraite qui correspond à la fonction **factorielle** dans le mini-langage. Appliquez ce terme à la valeur 10.

4.2 Extension du mini-langage

Étendez l'implantation du mini-langage avec la construction `for` que nous avons introduite dans la sémantique à la fin du TD. Prenez soin de définir toutes les fonctions auxiliaires nécessaires pour la construction et l'évaluation de la construction **for**. Ajoutez également les tests nécessaires pour vérifier le bon fonctionnement de cette sémantique de **for**.