Introduction à la compilation – TD 7 : Sémantique

Université Paris Diderot – Licence 3

(2011-2012)

Nous considérons un langage de programmation impératif simple dont la syntaxe est spécifiée par la grammaire suivante :

e	::=		**Expression**
	|	n	Entier
	|	x	Variable
	|	$e \quad e$	Opération binaire
c	::=		**Commande**
	|	$x := e$	Affectation
	|	**skip**	Commande vide
	|	$c; c$	Séquencement
	|	**while** e **do** c	Boucle non bornée
	|	**if** e **then** c **else** c	Branchement

L'opérateur dénote une opération binaire parmi $+$, $-$, $/$, et .

Exercice 1 (Syntaxe)

1. *À votre avis, comment écrire le calcul de* $5!$ *dans ce langage ?*

   Réponse: En supposant que l'entier 0 code le faux et les autres entiers le vrai alors :

   $x := 1;$
   $n := 5;$
   while n **do** (
   $x := n \quad x;$
   $n := n - 1;$
)

   est un programme dont l'évaluation calcule $5!$ dans la variable x.

2. *Proposez un type OCaml des arbres de syntaxe abstraite correspondant à la grammaire définie plus haut.*

   Réponse:

```
type binary_operator = int → int → int

type identifier = string

type expression =
  | Int of int
  | Binop of binary_operator * expression * expression
  | Variable of identifier

type command =
  | Assign of identifier * expression
```

```
  | Skip
  | Seq of commands
  | If of expression * command * command
  | While of expression * command

and commands = command list
```

3. *Donnez l'expression OCaml correspondant au programme de la question 1.*
   Réponse:

```
let fact5 =
  Seq [
    Assign ("x", Int 1);
    Assign ("n", Int 5);
    While (Variable "n", Seq [
      Assign ("x", Binop (( * ), Variable "n", Variable "x"));
      Assign ("n", Binop (( - ), Variable "n", Int 1));
    ])
  ]
```

□

Ce langage est impératif : la sémantique d'un programme est la transformation d'un état initial, formé d'une mémoire, en un nouvel état, dans lequel les valeurs associées aux variables peuvent avoir été modifiées et de nouvelles variables ont pu être allouées. On modélise une mémoire de la même façon qu'un environnement, à l'aide d'un dictionnaire M dont la syntaxe est :

			Mémoire
M	$::=$		
	$\mid$	$\bullet$	Mémoire vide
	$\mid$	$M + \{x \to n\}$	Mémoire où la variable x vaut n

Exercice 2 (Mémoire)

1. *En vous inspirant de définitions du cours, donnez les règles du jugement « $M(x) = n$ » qui se lit « Dans la mémoire M, la variable x a la valeur n. ».*
   Réponse:

$$\frac{}{(M + \{x \to n\})(x) = n} \qquad \frac{M(x) = n \qquad x = y}{(M + \{y \to n'\})(x) = n}$$

□

Exercice 3 (Sémantique des expressions)

1. *Donnez la sémantique à grands pas des expressions en donnant les règles d'un jugement « $M \quad e \quad n$ » qui se lit « Dans la mémoire M, l'expression e s'évalue en l'entier n ».*
   Réponse: Voir le cours.
2. *Donnez la sémantique à petit pas des expressions en donnant les règles d'un jugement « $M \quad e \to e'$ » qui se lit « Dans la mémoire M, l'expression e se réduit en un pas en l'expression e' ». Parmis ces règles, lesquelles sont des règles de réduction, lesquelles sont des règles de passage au contexte ?*
   Réponse: Voir le cours.

□

Exercice 4 (Sémantique à grands pas des commandes)

1. *Donnez les règles de sémantique à grands pas du jugement « $M \quad c \quad M'$ » qui se lit « La commande c transforme la mémoire M en la mémoire M'. ».*

   Réponse:

$$\frac{}{M \quad \textbf{skip} \quad M} \qquad \frac{M \quad e \quad n}{M \quad x := e \quad M + \{x \to n\}} \qquad \frac{M \quad c_1 \quad M_1 \qquad M \quad c_2 \quad M_2}{M \quad c_1; c_2 \quad M_2}$$

$$\frac{M \quad e \quad 0}{M \quad \textbf{while } e \textbf{ do } c \quad M} \qquad \frac{M \quad e \quad n \qquad n = 0 \qquad M \quad c; \textbf{ while } e \textbf{ do } c \quad M'}{M \quad \textbf{while } e \textbf{ do } c \quad M'}$$

$$\frac{M \quad e \quad n \qquad n = 0 \qquad M \quad c_1 \quad M'}{M \quad \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2 \quad M'} \qquad \frac{M \quad e \quad 0 \qquad M \quad c_2 \quad M'}{M \quad \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2 \quad M'}$$

2. *Donnez le programme OCaml correspondant à l'interprète de ce langage.*

   Réponse:

```
let read_memory x mem = List.assoc x mem
let write_memory x v mem = (x, v) :: mem

let rec exp mem = function
  | Int n → n
  | Variable x → read_memory x mem
  | Binop (f, e1, e2) → f (exp mem e1) (exp mem e2)

let rec command mem = function
  | Assign (x, e) →
    write_memory x (exp mem e) mem
  | Skip →
    mem
  | Seq cs →
    List.fold_left command mem cs
  | (While (e, b)) as c →
    if exp mem e = 0 then mem else command mem (Seq [b ; c])
  | If (e, c1, c2) →
    if exp mem e = 0 then command mem c1 else command mem c2
```

□

Pour définir une sémantique à petits pas pour ce langage, on introduit une notion de continuation K, représentant la suite du calcul pour un état donné du programme. La grammaire des continuations est :

K	$::=$		**Continuation**
		halt	Fin du calcul
	|	$c; K$	La prochaine commande est c

La sémantique à petits pas des commandes est une relation de réécriture qui transforme une configuration formée d'une mémoire M et d'une continuation K en une nouvelle configuration. La configuration initiale est $(\bullet, c; \textbf{halt})$ où c est la commande à évaluer. Une configuration est finale si elle est de la forme $(M, \textbf{halt})$.

Exercice 5 (Sémantique à petits pas des commandes)

1. *Donnez les règles de sémantique à petits pas qui définissent le jugement « $(M, K) \to (M', K')$ » qui se lit « La configuration (M, K) se réduit en un pas d'évaluation en la configuration (M', K'). ». Pour vous simplifier la tâche, vous pouvez vous appuyez sur le jugement de sémantique à grands pas des expressions. Quelles sont les règles de passage au contexte ? Quelles sont les règles de réduction ?*

   Réponse:

$$\frac{M \quad e \quad n}{(M, x := e; K) \to (M + \{x \to n\}, K)} \qquad \frac{}{(M, \textbf{skip}; K) \to (M, K)} \qquad \frac{}{(M, (c_1; c_2); K) \to (M, c_1; (c_2; K))}$$

$$\frac{M \quad e \quad 0}{(M, \textbf{while } e \textbf{ do } c; K) \to (M, K)} \qquad \frac{M \quad e \quad n \qquad n = 0}{(M, \textbf{while } e \textbf{ do } c; K) \to (M, c; (\textbf{while } e \textbf{ do } c; K))}$$

$$\frac{M \quad e \quad 0}{(M, \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2; K) \to (M, c_2; K)} \qquad \frac{M \quad e \quad n \qquad n = 0}{(M, \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2; K) \to (M, c_1; K)}$$

   Il n'y a que des règles de réduction.

2. *Donnez le code OCaml pour l'interprète de cette sémantique à petits pas.*

   Réponse:

```
type cont =
  | Halt
  | Cont of command * cont

type configuration = memory * cont

let rec command : configuration → command → configuration =
  fun (mem, cont) → function
    | Skip → (mem, cont)
    | Assign (x, e) → (write_memory x (exp mem e) mem, cont)
    | Seq [] → (mem, cont)
    | Seq (c :: cs) → (mem, Cont (c, Cont (Seq cs, cont)))
    | While (e, b) →
      if exp mem e = 0 then
        (mem, cont)
      else
        (mem, Cont (b, Cont (While (e, b), cont)))
    | If (e, c1, c2) →
      if exp mem e = 0 then
        (mem, Cont (c2, cont))
      else
        (mem, Cont (c1, cont))

let initial_configuration c = ([], Cont (c, Halt))

let rec eval = function
  | (m, Halt) → m
  | (m, Cont (c, cont)) → eval (command (m, cont) c)

let eval c = eval (initial_configuration c)
```

□