Examen du cours PROG1

6 novembre 2023

Durée : 1 heure 30 minutes.
Documents autorisés, machines interdites.

1 Sémantique de Mini-ML

Les règles de la sémantique de Mini-ML vue en cours sont rappelées en Figure 1 (on omet la construction **fixfun** inutile ici). On considère dans cet exercice diverses modifications de cette sémantique. Chaque question porte sur une modification indépendante des autres, et à chaque fois deux choses sont demandées :

- Commenter brièvement (en un paragraphe) le sens de la modification de la sémantique : expliquer informellement ce que dit / ce que fait la nouvelle règle, en reliant éventuellement avec des concepts généraux des langages de programmation.
- Indiquer si la modification affecte la relation d'évaluation définie : ce qui était dérivable le reste-t-il ? de nouvelles évaluations sont elles possibles ? sont-elles désirables/correctes ? Aucune preuve n'est attendue ici ; un paragraphe devrait suffire.

Question 1 On considère l'ajout de la règle suivante :

$$\frac{E \vdash \sigma, M \Downarrow \sigma_1, \langle E', \mathbf{fun}\ x \mapsto M' \rangle \quad E' \vdash \sigma_1, M' \Downarrow \sigma', R}{E \vdash \sigma, (M\ N) \Downarrow \sigma', R}\ x \notin \mathrm{dom}(E)$$

Question 2 On remplace la règle d'évaluation pour les fonctions par

$$\frac{}{E \vdash \sigma, (\mathbf{fun}\ x \mapsto M) \Downarrow \sigma, \langle E', \mathbf{fun}\ x \mapsto M \rangle}\ E' = E|_{\mathsf{fv}(\mathsf{fun}\ x \mapsto M)}$$

Question 3 On modifie la règle d'évaluation pour la séquence :

$$\frac{E \vdash \sigma, M \Downarrow \sigma_1, V \quad E \vdash \sigma_2, M' \Downarrow \sigma_3, R}{E \vdash \sigma, (M; M') \Downarrow \sigma_3, R}\ \sigma_2 = \sigma_1|_{\{A \in \mathcal{A}\ |\ E(x)=A \text{ pour un } x \in \mathsf{dom}(E)\}}$$

Expressions arithmétiques

$$\frac{}{E \vdash \sigma, n \Downarrow \sigma, n} \qquad \frac{E \vdash \sigma, M \Downarrow \sigma_1, m \quad E \vdash \sigma_1, N \Downarrow \sigma', n}{E \vdash \sigma, (M \oplus N) \Downarrow \sigma', (m+n)} \qquad \text{etc.}$$

$$\frac{E \vdash \sigma, M \Downarrow \sigma', 0 \quad E \vdash \sigma', M' \Downarrow \sigma'', R}{E \vdash \sigma, (\textbf{ifz } M \textbf{ then } M' \textbf{ else } M'') \Downarrow \sigma'', R} \qquad \frac{E \vdash \sigma, M \Downarrow \sigma', n \quad E \vdash \sigma', M'' \Downarrow \sigma'', R}{E \vdash \sigma, (\textbf{ifz } M \textbf{ then } M' \textbf{ else } M'') \Downarrow \sigma'', R}\ n \in \mathbb{Z}^*$$

Cœur fonctionnel

$$\frac{}{E \vdash \sigma, x \Downarrow \sigma, E(x)} \qquad \frac{}{E \vdash \sigma, (\textbf{fun } x \mapsto M) \Downarrow \sigma, \langle E, \textbf{fun } x \mapsto M \rangle}$$

$$\frac{E \vdash \sigma, M \Downarrow \sigma_1, \langle E', \textbf{fun } x \mapsto M' \rangle \quad E \vdash \sigma_1, N \Downarrow \sigma_2, V \quad E' + (x \mapsto V) \vdash \sigma_2, M' \Downarrow \sigma', R}{E \vdash \sigma, (M\ N) \Downarrow \sigma', R}$$

Séquençage

$$\frac{E \vdash \sigma, M \Downarrow \sigma', V \quad E + (x \mapsto V) \vdash \sigma', M' \Downarrow \sigma'', R}{E \vdash \sigma, \textbf{let } x = M \textbf{ in } M' \Downarrow \sigma'', R}$$

$$\frac{E \vdash \sigma, M \Downarrow \sigma', V \quad E \vdash \sigma', M' \Downarrow \sigma'', R}{E \vdash \sigma, (M; M') \Downarrow \sigma'', R} \qquad \frac{}{E \vdash \sigma, () \Downarrow \sigma, ()}$$

Références

$$\frac{E \vdash \sigma, M \Downarrow \sigma', V}{E \vdash \sigma, (\textbf{ref } M) \Downarrow \sigma'', A}\ A \notin \mathsf{dom}(\sigma'),\ \sigma'' = \sigma' + (A \mapsto V)$$

$$\frac{E \vdash \sigma, M \Downarrow \sigma', A}{E \vdash \sigma, !M \Downarrow \sigma', \sigma'(A)} \qquad \frac{E \vdash \sigma, M \Downarrow \sigma', A \quad E \vdash \sigma', M' \Downarrow \sigma'', V}{E \vdash \sigma, (M := M') \Downarrow \sigma'' + (A \mapsto V), R}$$

FIGURE 1 – Sémantique à grands pas de mini-ML

2 Notation post-fixe

On considère un langage d'expressions arithmétiques en notation post-fixe, construites sur un ensemble fini d'opérateurs binaires $\mathcal{O} \subseteq (\mathbb{N}^2 \to \mathbb{N})$:

$$E ::= n \mid EEf \qquad (n \in \mathbb{N}, f \in \mathcal{O})$$

Contrairement à notre habitude dans ce cours, cette grammaire spécifie ici la syntaxe *concrète* (et non abstraite) de notre langage d'expressions : nous ne verrons pas les expressions de E comme des arbres de syntaxe abstraite, mais comme des mots sur l'alphabet $\Sigma = \mathbb{N} \cup \mathcal{O}$. En particulier, l'usage des parenthèses est exclu. En OCaml, les expressions seront représentées dans le type suivant :

```
type binop = int -> int -> int
type token = Int of int | Op of binop
type expr = token list
```

Par exemple, si $\mathcal{O}$ contient les fonctions d'addition et de multiplication $+$ et $\times$, l'expression $123\times+$ est représentée par `Int 1 :: Int 2 :: Int 3 :: Op ( * ):: Op (+):: []`.

On définit la sémantique des expressions à petits pas, en les munissant d'une relation binaire $\to$ définie par $unmfv \to upv$ pour tous $u, v \in \Sigma^*$, $n, m \in \mathbb{N}$, $f \in \mathcal{O}$ et $p = f(n, m)$. Dans la suite, on écrira directement $unmfv \to uf(n, m)v$. Sur l'exemple précédent, on a ainsi $(123\times+) \to (16+) \to 7$.

Question 1 Démontrer que la relation $\to$ n'admet pas de réductions infinies, en bornant explicitement la longueur des réductions à partir d'une expression donnée.

Question 2 Démontrer que la relation $\to$ est confluente[1].

On définit la restriction $\to_L$ de $\to$: $unmfv \to_L uf(n, m)v$ pour tous $u \in \mathbb{N}^*$, $v \in \Sigma^*$, $n, m \in \mathbb{N}$, $f \in \mathcal{O}$ (remarquer que u n'est plus un mot sur Σ). On notera $\to_L^*$ la clôture réflexive transitive de $\to_L$, et $\to_L^+$ sa clôture transitive.

Question 3 Expliquer pourquoi $\to_L$ n'est pas seulement la restriction de $\to$ selon une stratégie de réduction le plus à gauche possible. Montrer qu'on a néanmoins, pour tous $u \in \Sigma^*$ et $n \in \mathbb{N}$, $u \to^* n$ ssi $u \to_L^* n$.

Question 4 Soient $n, k \in \mathbb{N}$, $f \in \mathcal{O}$ et $u, v \in \Sigma^*$. Montrer que $ku \to_L^+ nfv$ ssi il existe $p \in \mathbb{N}$, $g \in \mathcal{O}$ et $w \in \Sigma^*$ tels que $u \to_L^* pgw$ et $g(k, p)w \to_L^* nfv$.

On considère maintenant la fonction d'évaluation suivante en OCaml :

```
let rec eval : expr -> int = function
  | [Int n] -> n
  | Int n :: v ->
    let m,f,w = eval' v in
    eval (Int (f n m) :: w)
```

1. Pour tous u, v_1, v_2 tels que $v_1 \leftarrow^* u \to^* v_2$, il existe w tel que $v_1 \to^* w \leftarrow^* v_2$.

```
  | _ -> failwith "erreur"

and eval' : expr -> int * binop * expr = function
  | Int n :: Op f :: v -> n, f, v
  | Int k :: u ->
    let p,g,w = eval' u in
    eval' (Int (g k p) :: w)
  | _ -> failwith "erreur"
```

Question 5

(a) Montrer que `eval' u = (n,f,v)` ssi $u \to_L^* nfv$.

(b) Montrer que `eval u = n` ssi $u \to_L^* n$.

Question 6 Réécrire les fonctions d'évaluation en style par passage de continuation.

Question 7 Proposer une version récursive terminale des fonctions d'évaluation, qui n'utilise pas l'ordre supérieur : il n'est donc plus question de passer une continuation en argument ; plus généralement, aucune clôture ne doit être allouée pendant l'exécution du programme, les seules fonctions autorisées sont celles définies initialement à toplevel. Il n'est pas exigé que vos nouvelles fonctions soient obtenues en transformant les fonctions précédentes (par défonctionnalisation) mais procédez ainsi si possible, en expliquant les étapes non évidentes de la transformation.

3 Codages en λ-calcul

Dans cet exercice on considère le λ-calcul pur. On utilisera la β-réduction notée $\to_\beta$, sans stratégie. On rappelle les codages des booléens, des entiers et des paires :

$$\begin{aligned} \top &\stackrel{\text{def}}{=} \lambda x.\lambda y.\ x \\ \bot &\stackrel{\text{def}}{=} \lambda x.\lambda y.\ y \\ \overline{n} &\stackrel{\text{def}}{=} \lambda f.\lambda x.\ f^n\ x \quad (\text{pour } n \in \mathbb{N}) \\ \langle M, N\rangle &\stackrel{\text{def}}{=} \lambda f.\ f\ M\ N \end{aligned}$$

On se donne la fonction prédecesseur : on pourra utiliser le terme Pred en supposant que, pour tout $n > 0$, Pred $\overline{n} \to_\beta^* \overline{n-1}$. On se donnera de même les codages Succ et Mult pour le successeur et la multiplication. Tout autre codage devra être explicité.

Question 1 Donner un λ-terme Swap qui code l'échange des composantes d'une paire : pour tous M, N on veut avoir Swap $\langle M, N\rangle \to_\beta^* \langle N, M\rangle$.

Question 2 Donner des λ-termes codant les opérations suivantes sur les entiers :

a. Leq pour l'ordre non-strict : Leq $\overline{n}\ \overline{m} \to_\beta^* \top$ si $n \leq m$ et $\bot$ sinon ;

b. Eq pour l'égalité.

Question 3 Donner un λ-terme Foldi implémentant l'itération sur un entier : pour tous F, M_0 et n, pour tous $(M_k)_{0<k\leq n}$ tels que $F\ M_k\ \overline{k} \to_\beta^* M_{k+1}$, on veut Foldi $F\ \overline{n}\ M_0 \to_\beta^* M_n$.

Question 4 Donner un λ-terme Div qui code la divisibilité : Div $\overline{n}\ \overline{m} \to_\beta^* \top$ si n divise m et $\bot$ sinon.