Sémantique des langages de programmation
Examen final

Semestre 5, 2012-2013

Durée : 1h45.
Tout document personnel autorisé (pas de prêt entre voisins).
Ce sujet comporte deux exercices indépendants, qui peuvent être traités dans l'ordre voulu. Il contient 4 pages. Il va de soi que toute réponse devra être justifiée.

Exercice 1 : Couples

On étend la syntaxe du langage MiniML vu en cours avec le possibilité de créer des couples et de récupérer les membres d'un couple (projections) :

$$e ::= n \mid x \mid \texttt{fun}\ x\ \texttt{->}\ e \mid e\ e \mid e + e \mid \mathbf{(e, e)} \mid \mathtt{fst}(\boldsymbol{e}) \mid \mathtt{snd}(\boldsymbol{e})$$

Sémantique

1. Étendre la sémantique à grand pas par valeur de MiniML (rappelée figure 1) pour donner une sémantique aux couples et projections. Informellement, un couple d'expressions est évalué en le couple des valeurs des expressions. La première projection ($\texttt{fst}(e)$) d'un couple s'évalue comme le membre gauche du couple, la deuxième ($\texttt{snd}(e)$) comme le membre droit.
2. Comment la définition des valeurs doit-elle être étendue ?
3. Donner la sémantique à grand pas par valeur des expressions suivantes :

$$(1 + 2, (\texttt{fun}\ x\ \texttt{->}\ 3)\ 4)$$
$$(\texttt{fun}\ f\ \texttt{->}\ (f\ 1, f\ (\texttt{fun}\ x\ \texttt{->}\ x)))\ (\texttt{fun}\ y\ \texttt{->}\ y)$$
$$\texttt{snd}((\texttt{fun}\ x\ \texttt{->}\ (x, x))\ 2)$$
$$(\texttt{fun}\ c\ \texttt{->}\ (c, \texttt{fst}(c)))\ (3 + 1, 8)$$

4. Quelle condition (syntaxique) sur l'expression e implique l'équivalence des programmes e et $(\texttt{fst}(e), \texttt{snd}(e))$? Détaillez votre réponse.
5. Étendre la sémantique à petit pas par valeur de MiniML (rappelée figure 2) pour donner une sémantique aux couples et projections.
6. Démontrer l'équivalence des sémantiques opérationnelles à grands pas et à petit pas. On pourra se contenter d'exposer les nouveaux cas.

$$(F_1)\frac{}{n \to_v n} \qquad (F_2)\frac{}{\texttt{fun}\ x\ \texttt{->}\ e \to_v \texttt{fun}\ x\ \texttt{->}\ e}$$

$$(F_{v3})\frac{e_1 \to_v \texttt{fun}\ x\ \texttt{->}\ e \qquad e_2 \to_v v \qquad \{v/x\}e \to_v v'}{e_1\ e_2 \to_v v'}$$

$$(F_4)\frac{e_1 \to_v v_1 \qquad e_2 \to_v v_2}{e_1 + e_2 \to_v v_1 [\![+]\!] v_2}$$

FIGURE 1: Sémantique opérationnelle à grand pas par valeur de MiniML

$$(FS_{v1})\frac{}{(\texttt{fun}\ x\ \texttt{->}\ e)v \hookrightarrow_v \{v/x\}e}$$

$$(FS_{v2})\frac{e_1 \hookrightarrow_v e_1'}{e_1\ e_2 \hookrightarrow_v e_1'\ e_2} \qquad (FS_{v3})\frac{e_2 \hookrightarrow_v e_2'}{v\ e_2 \hookrightarrow_v v\ e_2'}$$

$$(FS_{v4})\frac{}{v_1 + v_2 \hookrightarrow_v v_1 [\![+]\!] v_2}$$

$$(FS_{v5})\frac{e_1 \hookrightarrow_v e_1'}{e_1 + e_2 \hookrightarrow_v e_1' + e_2} \qquad (FS_{v6})\frac{e_2 \hookrightarrow_v e_2'}{v + e_2 \hookrightarrow_v v + e_2'}$$

FIGURE 2: Sémantique opérationnelle à petit pas par valeur de MiniML

$$\text{CST\_\_ENT}\ \frac{}{\Gamma \vdash n : int}\ n \in \mathbb{Z} \qquad \text{ID}\ \frac{}{\Gamma \vdash \texttt{x} : \tau}\ \texttt{x} : \tau \in \Gamma$$

$$\text{ABS}\ \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \texttt{fun}\ x\ \texttt{->}\ e : \tau_1 \to \tau_2} \qquad \text{APP}\ \frac{\Gamma \vdash e : \tau_1 \to \tau_2 \qquad \Gamma \vdash e' : \tau_1}{\Gamma \vdash e\ e' : \tau_2}$$

$$\text{OP}\ \frac{\Gamma \vdash e_1 : int \qquad \Gamma \vdash e_2 : int}{\Gamma \vdash e_1 + e_2 : int}$$

FIGURE 3: Système de type monomorphe de MiniML

Typage

On étend également la syntaxe des types pour prendre en compte les couples. Pour les types monomorphes on obtient :

$$\tau ::= int \mid \alpha \mid \tau \to \tau \mid \boldsymbol{\tau * \tau}$$

7. Étendre le système de type monomorphe de MiniML (rappelé figure 3) pour prendre en compte les couples et les projections.
8. Les expressions suivantes sont-elles typables ? Si oui, donner le type principal et une dérivation de typage, si non, expliquer pourquoi :

$$(1 + 2, (\texttt{fun}\ x\ \texttt{->}\ 3)\ 4)$$
$$(\texttt{fun}\ f\ \texttt{->}\ (f\ 1, f\ (\texttt{fun}\ x\ \texttt{->}\ x)))\ (\texttt{fun}\ y\ \texttt{->}\ y)$$
$$\texttt{snd}((\texttt{fun}\ x\ \texttt{->}\ (x, x))\ 2)$$
$$(\texttt{fun}\ c\ \texttt{->}\ (c, \texttt{fst}(c)))\ (3 + 1, 8)$$

9. On ajoute la construction `let` et on considère des types polymorphes. Quelles modifications faut-il apporter au système de type ? L'expression

$$\texttt{let}\ f\ \texttt{=}\ \texttt{fun}\ y\ \texttt{->}\ y\ \texttt{in}\ (f\ 1, f\ (\texttt{fun}\ x\ \texttt{->}\ x))$$

est-elle typable ? Justifiez votre réponse.

10. Étendre l'algorithme d'inférence de type de Hindley-Milner (cf. figure 4). Appliquer le à l'expression de la question précédente en détaillant les étapes.

$$
\begin{aligned}
W(\Gamma, n) =& (int, \emptyset) && n \in \mathbb{Z} \\
W(\Gamma, x) =& (triv(\Gamma(x)), \emptyset) \\
W(\Gamma, \texttt{fun } x \texttt{ -> } e) =& \text{soit } \alpha = \text{une variable fraîche} \\
& \quad \text{soit } (\tau, s) = W((\Gamma, x : \alpha), e) \\
& \quad (s\alpha \to \tau, s) \\
W(\Gamma, e_1\ e_2) =& \text{soit } (\tau_1, s_1) = W(\Gamma, e_1) \\
& \quad \text{soit } (\tau_2, s_2) = W(s_1(\Gamma), e_2) \\
& \quad \text{soit } \alpha = \text{une variable fraîche} \\
& \quad \text{soit } \mu = mgu(s_2\tau_1, \tau_2 \to \alpha) \\
& \quad (\mu\alpha, \mu \circ s_2 \circ s_1) \\
W(\Gamma, \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2) =& \text{soit } (\tau_1, s_1) = W(\Gamma, e_1) \\
& \quad \text{soit } (\tau_2, s_2) = W((s_1(\Gamma), x : Gen(\tau_1, \Gamma)), e_2) \\
& \quad \tau_2, s_2 \circ s_1
\end{aligned}
$$

FIGURE 4: Algorithme W de Hindley-Milner

Exercice 2 : Variables locales

On étend la syntaxe de IMP pour y ajouter des blocs qui peuvent contenir des déclarations locales de variables.

$$c ::= \mathsf{skip} \mid x := a \mid c_1; c_2 \mid \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ b\ \mathsf{do}\ c \mid \textbf{var } \boldsymbol{x} \mid \textbf{begin } \boldsymbol{c} \textbf{ end}$$

Informellement, la portée d'une variable est située entre l'endroit où elle est déclarée et la fin du bloc contenant sa déclaration, sauf si une variable de même nom est déclarée dans un bloc imbriqué. Par exemple, dans le programme

```
1  begin
2    var y;
3    begin
4       y := 1;
5       var x;
6       x := y + 2;
7       begin
8         y := y - x;
9         var x;
10        x := y
11      end;
12      x := 2
13   end;
14   y := y - 1
15 end
```

$$(A_1)\frac{}{\langle n, \sigma\rangle \rightsquigarrow n}\ (n \in \mathbb{Z}) \qquad (A_2)\frac{}{\langle x, \sigma\rangle \rightsquigarrow \sigma(x)}\ (x \in V)$$

$$(A_3)\frac{\langle a_1, \sigma\rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma\rangle \rightsquigarrow n_2}{\langle a_1 + a_2, \sigma\rangle \rightsquigarrow n_1 [\![+]\!] n_2} \quad (A_4)\frac{\langle a_1, \sigma\rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma\rangle \rightsquigarrow n_2}{\langle a_1 \times a_2, \sigma\rangle \rightsquigarrow n_1 [\![\times]\!] n_2}$$

$$(A_5)\frac{\langle a_1, \sigma\rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma\rangle \rightsquigarrow n_2}{\langle a_1 - a_2, \sigma\rangle \rightsquigarrow n_1 [\![-]\!] n_2} \quad (A_6)\frac{\langle a_1, \sigma\rangle \rightsquigarrow n_1 \quad \langle a_2, \sigma\rangle \rightsquigarrow n_2}{\langle a_1 / a_2, \sigma\rangle \rightsquigarrow n_1 [\![/]\!] n_2}$$

$$(C_1)\frac{}{\langle \mathsf{skip}, \sigma\rangle \to \sigma} \qquad (C_2)\frac{\langle a, \sigma\rangle \rightsquigarrow n}{\langle x := a, \sigma\rangle \to \sigma[x \leftarrow n]} \quad n \in \mathbb{Z}$$

$$(C_3)\frac{\langle c_1, \sigma\rangle \to \sigma_1 \quad \langle c_2, \sigma_1\rangle \to \sigma_2}{\langle c_1; c_2, \sigma\rangle \to \sigma_2}$$

$$(C_4)\frac{\langle b, \sigma\rangle \rightsquigarrow \mathsf{true} \quad \langle c_1, \sigma\rangle \to \sigma_1}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, \sigma\rangle \to \sigma_1} \quad (C_5)\frac{\langle b, \sigma\rangle \rightsquigarrow \mathsf{false} \quad \langle c_2, \sigma\rangle \to \sigma_2}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, \sigma\rangle \to \sigma_2}$$

$$(C_6)\frac{\langle b, \sigma\rangle \rightsquigarrow \mathsf{false}}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \to \sigma}$$

$$(C_7)\frac{\langle b, \sigma\rangle \rightsquigarrow \mathsf{true} \quad \langle c, \sigma\rangle \to \sigma_1 \quad \langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma_1\rangle \to \sigma_2}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \to \sigma_2}$$

FIGURE 5: Sémantique opérationnelle à grand pas de IMP (sans les expressions booléennes)

la portée de `x` déclarée ligne 5 est de la ligne 5 à la ligne 8 et de la ligne 12 à la ligne 13. On dira qu'un programme est bien défini si un bloc contient au plus une seule déclaration d'une même variable (mais deux blocs imbriqués peuvent déclarer la même variable). On supposera par la suite que tous les programmes considérés sont bien définis.

Pour définir la sémantique du langage étendu, on utilisera comme environnement des piles de valuations notées $\sigma_1 \triangleright \cdots \triangleright \sigma_n$. σ_n sera la valuation pour les variables déclarées dans le bloc courant, σ_{n-1} celle pour le bloc qui l'entoure, etc. La sémantique opérationnelle à grand pas définira donc une relation entre un couple formé d'un programme et d'une telle pile, et une pile : $\langle c, \sigma_1 \triangleright \cdots \triangleright \sigma_n\rangle \to \sigma'_1 \triangleright \cdots \triangleright \sigma'_n$.

Informellement, la sémantique devient la suivante : la déclaration d'une variable initialise cette variable à 0 dans la valuation la plus à droite de la pile ; la valeur d'une variable est celle dans la valuation la plus à droite dans laquelle elle est définie ; de même, une variable est affectée dans la valuation la plus à droite où elle est déjà définie ; lors de l'entrée dans un bloc, une valuation vide est ajoutée à droite de la pile, et elle est dépilée à la sortie du bloc.

1. Modifier la sémantique opérationnelle à grand pas de IMP (cf. figure 5) pour prendre en compte ces changements. En particulier, il faudra modifier les règles (A_2) et (C_2) et ajouter des règles pour `var` x et `begin` c `end`.
2. Quelle est la sémantique du programme donné au début de l'exercice en partant de la pile vide ? Vous détaillerez l'arbre de dérivation le plus clairement possible (avec sa conclusion en bas et les noms des règles clairement mentionnées).
3. Les programmes c et `begin` c `end` sont-ils équivalents ? Justifier.