Analyse Syntaxique et Compilation, TD n° 4

Nous considérons un langage de programmation impérative simple. Nous utilisons une *mémoire* qui associe aux variables des valeurs (comme l'environnement utilisé en cours). Le jugement « $\eta(x) = n$ » pour une mémoire η, une variable x et une valeur n (entière) est défini par :

$$\frac{}{(\eta; (x \mapsto n))(x) = n} \qquad \frac{\eta(x) = n \qquad x \neq y}{(\eta; (y \mapsto n'))(x) = n}$$

On considère la sémantique opérationnelle à grands pas pour des expressions arithmétiques donnée par les règles suivantes ($\oplus$ dénote $+, -, /$ ou $*$). La valeur d'une expression dépend de la mémoire η.

$$\frac{\eta \vdash e_1 \Downarrow n_1 \qquad \eta \vdash e_2 \Downarrow n_2}{\eta \vdash e_1 \oplus e_2 \Downarrow n_1 \oplus_{\mathbb{N}} n_2} \qquad \frac{}{\eta \vdash n \Downarrow n} \qquad \frac{\eta(x) = n}{\eta \vdash x \Downarrow n}$$

Les règles de sémantique à petits pas sont données par :

$$\frac{(e_1, \eta) \to (e_1', \eta')}{(e_1 \oplus e_2, \eta) \to (e_1' \oplus e_2, \eta')} \qquad \frac{(e_2, \eta) \to (e_2', \eta')}{(n_1 \oplus e_2, \eta) \to (n_1 \oplus e_2', \eta')}$$

$$\frac{}{(n_1 \oplus n_2, \eta) \to (m, \eta)} \text{ avec } m = n_1 \oplus_{\mathbb{N}} n_2 \qquad \frac{}{(x, \eta) \to (\eta(x), \eta)} \text{ si } \eta(x) \text{ est défini}$$

Exercice 1 :

- Dérivez le jugement $\bullet; (x \mapsto 42); (y \mapsto 10) \vdash x + y * 2 \Downarrow 62$.
- Donnez la suite des pas de la sémantique à petits pas qui mène $x + y * 2$ à 62 avec la mémoire $\bullet; (x \mapsto 42); (y \mapsto 10)$
- Que se passe-t-il avec le terme $x + y$ avec la mémoire $\bullet; (y \mapsto 42)$?

Nous ajoutons les constantes **true**, **false** ainsi que $x \ \mathcal{R}^? y$, avec $\mathcal{R} \in \{<, >, \leq, \geq, =\}$. Leur sémantique est donnée par

$$\frac{}{\eta \vdash \textbf{false} \Downarrow \textbf{false}} \qquad \frac{}{\eta \vdash \textbf{true} \Downarrow \textbf{true}} \qquad \frac{\eta \vdash e_1 \Downarrow n_1 \qquad \eta \vdash e_2 \Downarrow n_2}{\eta \vdash e_1 \mathcal{R}^? e_2 \Downarrow \textbf{true}} \text{ si } n_1 \mathcal{R} n_2 \text{ est vrai}$$

$$\frac{\eta \vdash e_1 \Downarrow n_1 \qquad \eta \vdash e_2 \Downarrow n_2}{\eta \vdash e_1 \mathcal{R}^? e_2 \Downarrow \textbf{false}} \text{ si } n_1 \mathcal{R} n_2 \text{ est faux}$$

Nous considérons les programmes formés de commandes de la forme (1) **skip**, (2) $C_1; C_2$, (3) **if** e **then** C_1 **else** C_2, où e est une expression, et (4) les affectations de la forme $x := e$ où x est une variable et e une expression. La sémantique de l'affectation est donnée par

$$\frac{\eta \vdash e \Downarrow n}{\eta \vdash x := e \Downarrow \eta; (x \mapsto n)}$$

L'affectation transforme donc une mémoire vers une autre mémoire.

Exercice 2 :

- Donnez les règles de sémantique à grands pas pour la composition séquentielle ;
- Donnez les règles pour **if then else**

Nous ajoutons la commande **while** e **do** C. Sa sémantique est donnée par :

$$\frac{\eta \vdash e \Downarrow \textbf{false}}{\eta \vdash \textbf{while e do C} \Downarrow \eta} \qquad \frac{\eta \vdash e \Downarrow \textbf{true} \quad \eta \vdash C \Downarrow \eta' \quad \eta' \vdash \textbf{while e do C} \Downarrow \eta''}{\eta \vdash \textbf{while e do C} \Downarrow \eta''}$$

Exercice 3 :

- Évaluez $x := 2;\ y := 0;$ **while** $x > 0$ **do** $(y := y + x;\ x := x - 1)$
- Donnez les règles de la sémantique pour une commande de la forme **repeat** C **until** e

Exercice 4 :

Donnez des règles de la sémantique à petits pas pour les commandes. Les jugements sont de la forme $(C, \eta) \to (C', \eta')$. Par exemple les règles pour l'affectation (où n est une valeur) sont

$$\frac{(e, \eta) \to (e', \eta')}{(x := e, \eta) \to (x := e', \eta')} \qquad \frac{}{(x := n, \eta) \to (\textbf{skip}, \eta; (x \mapsto n)),}$$

La commande **while** peut être transformée en utilisant **if-then-else** et **skip**.

Exercice 5 :

On souhaite ajouter une commande **break**. Sa sémantique est donner informellement comme suit : Il y a deux modes d'exécution : un mode "normal" et un mode "arrêt". On passe du mode "normal" au mode "arrêt" lorsqu'on exécute un **break**. On passe du mode "arrêt" au mode "normal" lorsqu'on termine une itération de boucle **while** en sortant effectivement de la boucle. En mode "arrêt", on n'exécute aucune commande tant qu'on n'a pas terminé l'itération courante.

Pour tenir compte des deux modes de calcul on ajoute à la mémoire une composante.

On change le jugement d'évaluation d'une commande en rajoutant en sortie une composante valant **ok** si l'exécution est en mode « normal » et **stop** si elle est en mode « arrêt ». Le jugement devient donc de la forme :

$$\eta \vdash C \Downarrow (\eta', \text{m}) \text{ avec m} ::= \textbf{stop} \mid \textbf{ok}$$

- En déduire les nouvelles règles de sémantique à grands pas des commandes du langage.