Master mention informatique
Spécialité STL

MI030 – Analyse des programmes et sémantique (APS)
Deuxième examen réparti
Lundi 23 mai 2011, 13h30 – 15h30

Directives

1. Le contrôle dure 2h00.
2. Tous les documents sont autorisés.
3. Tous les appareils électroniques sont **prohibés** (y compris les téléphones portables, les assistants numériques personnels et les agendas électroniques).

Question 1. En vous inspirant des structures de données fonctionnelles pour les paires et les listes vues en TD, définissez la structure de données fonctionelle pile (« *stack* ») capable de répondre aux commandes suivantes : (5 points)

Soit p une pile :

$$\begin{aligned} (p\ \texttt{top}) &\Rightarrow\ t \\ (p\ \texttt{pop}) &\Rightarrow\ \langle t, p' \rangle \\ ((p\ \texttt{push})\ t') &\Rightarrow\ p'' \\ (p\ \texttt{isEmpty}) &\Rightarrow\ false \end{aligned}$$

où t la valeur au sommet de la pile p courante, p' la pile p dont on a retiré le sommet et p'' la pile p à laquelle a été ajoutée la valeur t' au sommet.

Solution :

$$\begin{aligned} \textit{make-stack} = (&\textbf{fix}\ \lambda f. \\ &\lambda t.\lambda s. \\ &\quad \lambda cmd. \\ &\quad\quad \textbf{if}\ (=\ cmd\ \texttt{top})\ \textbf{then}\ t \\ &\quad\quad \textbf{else if}\ (=\ cmd\ \texttt{pop})\ \textbf{then}\ \langle t, s \rangle \\ &\quad\quad \textbf{else if}\ (=\ cmd\ \texttt{push})\ \lambda v.((f\ v)\ ((f\ t)\ s)) \\ &\quad\quad \textbf{else if}\ (=\ cmd\ \texttt{isEmpty})\ false \\ &\quad\quad \textbf{else}\ erreur) \\ \textit{emptyStack} = (&\textbf{fix}\ \lambda f. \\ &\quad \lambda cmd. \\ &\quad\quad \textbf{if}\ (=\ cmd\ \texttt{push})\ \lambda v.((\textit{make-stack}\ v)\ f) \\ &\quad\quad \textbf{else if}\ (=\ cmd\ \texttt{isEmpty})\ true \\ &\quad\quad \textbf{else}\ erreur) \end{aligned}$$

Question 2. Introduisez les variables statiques (de classe) dans BOPL et dans sa sémantique dénotationnelle. À partir des modifications suivantes à sa syntaxe abstraite : (15 points)

La déclaration des classes est modifiée pour introduire les variables statiques :

$$c \quad ::= \quad \texttt{class}\ id\ ce\ v_1^*\ v_2^*\ m^*$$

où v_1^* représente les variables statiques de la classe, alors que v_2^* représente les variables d'instance (qui existaient déjà).

Et on introduit une instruction pour l'affectation de valeurs aux variables statiques de même qu'un expression pour la lecture de leurs valeurs :

$$\begin{array}{rcl} i & ::= & \ldots \mid \texttt{writeStaticField}\ ce\ id\ e \\ e & ::= & \ldots \mid \texttt{readStaticField}\ ce\ id \end{array}$$

où *ce* est l'expression de classe dénotant la classe dans laquelle on doit lire ou écrire dans la variable de nom *ide* ; l'expression e donne la valeur à affecter dans le `writeStaticField`.

(a) Décrivez en cinq lignes, en français, toutes les modifications à faire par rapport à la sémantique dénotationnelle de BOPL vue en cours pour introduire ces variables statiques.

(b) Donnez toutes ces modifications.

Solution :

L'idée générale consiste à reprendre pour les classes la solution adoptée pour les objets : leur adjoindre un « *object map* » pour leurs variables statiques. Cet « *object map* » va contenir, comme il se doit, la correspondance entre les noms de variables statiques de la classe et les adresses de leurs valeurs en mêmoire. On ajoute ensuite deux nouveaux codes au *make-ClassWrapper* pour traiter l'écrituare et la lecture dans une variable statique, en s'inspirant de ce qui est fait pour les objets. La création des classes est adaptée pour traiter les déclarations de variables statiques, alors que la commande et l'expression pour l'écriture et la lecture dans une variable statique sont définies en allant chercher la classe dans l'environnement, en lui passant le code correspondant à l'opération à faire, puis en fournissant les paramètres nécessaires.

$$
\begin{array}{l}
\mathit{make\text{-}ClassWrapper} = \lambda ide.\lambda ide_1^*.\lambda ide_2^*.\lambda dict.\lambda\sigma. \\
\qquad\qquad \textbf{let } \langle omap, \sigma_1\rangle = ((\mathit{make\text{-}ObjectMap}\ ide_1^*)\ \sigma) \\
\qquad\qquad \textbf{in } \langle \lambda super.\lambda n. \\
\qquad\qquad\qquad \textbf{if } (=\ n\ 0) \textbf{ then} \\
\qquad\qquad\qquad\quad \lambda ide_1.\textbf{let } found = (dist\ ide_1) \\
\qquad\qquad\qquad\qquad\qquad \textbf{in if } is\mathbf{U}(found) \textbf{ then } ((super\ 0)\ ide_1) \\
\qquad\qquad\qquad\qquad\qquad\quad \textbf{else } out\mathbf{Method}(found) \\
\qquad\qquad\qquad \textbf{else if } (=\ n\ 1) \textbf{ then} \\
\qquad\qquad\qquad\quad (append\ ide_2^*\ (super\ 1)) \\
\qquad\qquad\qquad \textbf{else if } (=\ n\ 2) \textbf{ then} \\
\qquad\qquad\qquad\quad \lambda\sigma.(((\mathit{make\text{-}Object}\ ide)\ (append\ ide_2^*\ (super\ 1)))\ \sigma) \\
\qquad\qquad\qquad \textbf{else if } (=\ n\ 3) \textbf{ then} \\
\qquad\qquad\qquad\quad \lambda ide.\lambda\sigma. \\
\qquad\qquad\qquad\qquad \textbf{let } loc = (omap\ ide) \\
\qquad\qquad\qquad\qquad \textbf{in if } is\mathbf{U}(loc) \textbf{ then } in\mathbf{EV}(out\mathbf{RV}(\sigma\ loc)) \\
\qquad\qquad\qquad\qquad\quad \textbf{else } erreur \\
\qquad\qquad\qquad \textbf{else if } (=\ n\ 4) \textbf{ then} \\
\qquad\qquad\qquad\quad \lambda ide.\lambda sv.\lambda\sigma. \\
\qquad\qquad\qquad\qquad \textbf{let } loc = (omap\ ide) \\
\qquad\qquad\qquad\qquad \textbf{in if } is\mathbf{U}(loc) \textbf{ then } (((updateStore\ \sigma)\ loc)\ sv) \\
\qquad\qquad\qquad\qquad\quad \textbf{else } erreur \\
\qquad\qquad\qquad \textbf{else } erreur, \\
\qquad\qquad \sigma_1\rangle
\end{array}
$$

$$
\begin{array}{l}
\mathscr{K} : Class \to Env \to \Sigma \to Env \otimes \Sigma \\
\mathscr{K}[\![\texttt{class}\ id\ ce\ v_1^*\ v_2^*\ m^*]\!]\rho\sigma = \\
\qquad \textbf{let } super = out\mathbf{Class}(\rho\ \mathscr{CE}[\![ce]\!]) \\
\qquad \textbf{and } \langle cw, \sigma_1\rangle = (((((\mathit{make\text{-}ClassWrapper}\ \mathscr{I}[\![id]\!])\ \mathscr{V}^*[\![v_1^*]\!])\ \mathscr{V}^*[\![v_2^*]\!])\ \mathscr{M}^*[\![m^*]\!]super)\ \sigma) \\
\qquad \textbf{in } \langle (((extendEnv\ \rho)\ ide)\ in\mathbf{DV}(cw\ super)), \sigma_1\rangle
\end{array}
$$

$$
\begin{array}{l}
\mathscr{K}^* : Class^* \to Env \to \Sigma \to Env \otimes \Sigma \\
\mathscr{K}^*[\![c^*]\!]\rho\sigma = \ \ \textbf{let}* loop = \lambda f.\lambda c^*.\lambda\rho.\lambda\sigma \\
\qquad\qquad\qquad\qquad \textbf{if } \neg null(c^*) \\
\qquad\qquad\qquad\qquad \textbf{then let}\langle\rho_1, \sigma_1\rangle = \mathscr{K}[\![head(c^*)]\!]\rho\sigma \textbf{ in } (((f\ (tail\ c^*))\ \rho_1)\ \sigma_1) \\
\qquad\qquad\qquad\qquad \textbf{else } \langle\rho, \sigma\rangle \\
\qquad\qquad \textbf{in } ((((\textbf{fix } loop)\ c^*)\ \rho)\ \sigma)
\end{array}
$$

$$
\begin{array}{l}
\mathscr{P}[\![\texttt{program}\ c^*\ v^*\ i]\!] = \ \ \textbf{let}* ide^* = \mathscr{V}^*[\![v^*]\!] \\
\qquad\qquad\qquad\qquad \textbf{and } \rho = (((extendEnv\ emptyEnv)\ \mathscr{I}[\![\texttt{Object}]\!])\ in\mathbf{DV}_2(theObjectClass)) \\
\qquad\qquad\qquad\qquad \textbf{and } (\sigma, loc^*) = ((allocate*\ \ emptyStore)\ \sharp ide^*) \\
\qquad\qquad\qquad\qquad \textbf{and } \rho_1 = (((extendEnv*\ \ \rho)\ ide^*)\ in\mathbf{DV}_1^*(in\mathbf{LV}_1^*(loc^*))) \\
\qquad\qquad\qquad\qquad \textbf{and } \langle\rho_2, \sigma_1\rangle = \mathscr{K}^*[\![c^*]\!]\rho_1\sigma \\
\qquad\qquad\qquad\qquad \textbf{in } (((\mathscr{C}[\![i]\!]\ \rho_2)\ \sigma_1)\ emptyOut)
\end{array}
$$

Et pour la nouvelle instruction et la nouvelle expression, on a :

$$\mathscr{C}[\![\texttt{writeStaticField}\ ce\ id\ e]\!]\rho\sigma o =$$
$$\quad \textbf{let}\ \langle ev_1, \sigma_1, o_1\rangle = \mathscr{E}[\![e]\!]\rho\sigma o$$
$$\quad \textbf{and}\ \sigma_2 = ((((out\textbf{Class}(\rho\ \mathscr{CE}[\![ce]\!])\ 4)\ \mathscr{I}[\![id]\!])\ in\textbf{SV}(out\textbf{RV}(ev_1)))\ \sigma_1)$$
$$\quad \textbf{in}\ \langle \sigma_2, o_1\rangle$$

$$\mathscr{E}[\![\texttt{readStaticField}\ ce\ id]\!]\rho\sigma o =$$
$$\quad \textbf{let}\ ev_1 = (((out\textbf{Class}(\rho\ \mathscr{CE}[\![ce]\!])\ 3)\ \mathscr{I}[\![id]\!])\ \sigma)$$
$$\quad \textbf{in}\ \langle ev_1, \sigma, o\rangle$$

Fin du contrôle.