Programmation, partiel: sémantique de Lisp

À rendre pour le vendredi 05 décembre 2003
en version papier : Jean Goubault-Larrecq, LSV ;
ou électronique : `goubault@lsv.ens-cachan.fr`

Recommandations. Votre copie (papier ou électronique) devra être lisible et bien structurée. La note tiendra compte autant du fond que de la présentation. D'autre part, vous devrez justifier toutes vos affirmations, soit par une preuve soit par un contre-exemple. (Ces recommandations sont bien sûr valables dans tout devoir.)

Contexte. Dans le devoir, il sera fait plusieurs fois références à la sémantique de mini-Caml. On pourra se référer à `http://www.lsv.ens-cachan.fr/~goubault/cours.html`.

Lisp est l'un des premiers langages fonctionnel. Nous en considérons une version simplifiée et adaptée. La syntaxe des expressions est :

$$\begin{array}{rcll} e & ::= & x & \text{variables} \\ & | & (e\ e) & \text{application} \\ & | & (\texttt{lambda}\ (x)\ e) & \text{fonction} \\ & | & (\texttt{let}\ (x\ e)\ e) & \text{abréviation} \\ & | & e; e & \text{séquence} \\ & | & (\texttt{setq}\ x\ e) & \text{affectation} \end{array}$$

En voici la sémantique naturelle, qui permet de dériver des jugements de la forme $\rho \vdash e \Rightarrow \rho', V$, où ρ et ρ' sont des *environnements*, c'est-à-dire des applications qui à chaque variable associent des valeurs, et où V est une *valeur*, définie par la grammaire :

$$\begin{array}{rcll} V & ::= & n & \text{entier (dans } \mathbb{Z}) \\ & | & \langle x, e \rangle & \text{fonction définie} \end{array}$$

Les règles de la sémantique naturelle sont les suivantes :

$$\frac{}{\rho \vdash x \Rightarrow \rho, \rho(x)}\ (Var)$$

$$\frac{\rho \vdash e_1 \Rightarrow \rho_1, \langle x, e_3 \rangle \quad \rho_1 \vdash e_2 \Rightarrow \rho_2, V_2 \quad \rho_2[x := V_2] \vdash e_3 \Rightarrow \rho', V}{\rho \vdash (e_1\ e_2) \Rightarrow \rho'[x := \rho_2(x)], V}\ (App)$$

$$\frac{}{\rho \vdash (\texttt{lambda}\ (x)\ e) \Rightarrow \rho, \langle x, e \rangle}\ (Fun)$$

$$\frac{\rho \vdash e_1 \Rightarrow \rho_1, V_1 \quad \rho_1[x := V_1] \vdash e_2 \Rightarrow \rho_2, V_2}{\rho \vdash (\texttt{let}\ (x\ e_1)\ e_2) \Rightarrow \rho_2[x := \rho_1(x)], V_2}\ (Let)$$

$$\frac{\rho \vdash e_1 \Rightarrow \rho_1, V_1 \quad \rho_1 \vdash e_2 \Rightarrow \rho_2, V_2}{\rho \vdash e_1; e_2 \Rightarrow \rho_2, V_2}\ (Seq)$$

$$\frac{\rho \vdash e \Rightarrow \rho', V}{\rho \vdash (\texttt{setq}\ x\ e) \Rightarrow \rho'[x := V], V}\ (Assign)$$

1. Écrire la dérivation de sémantique naturelle la plus générale (dans un sens intuitif) pour les expressions :

$$(a)\ (\texttt{setq}\ x\ (\texttt{lambda}\ (y)\ y)); (x\ z) \qquad (b)\ (\texttt{let}\ (x\ (\texttt{lambda}\ (y)\ y))\ (x\ z))$$

   Quelle est la différence entre les deux expressions ?

2. Les constructions $(\texttt{let}\ (x\ e)\ e')$ et $((\texttt{lambda}\ (x)\ e')\ e)$ sont-elles équivalentes ? On dira que deux expressions e et e' sont *équivalentes* si et seulement si : $\rho \vdash e \Rightarrow \rho', V$ est dérivable si et seulement si $\rho \vdash e' \Rightarrow \rho', V$ l'est. Justifier.

3. On ajoute au langage une construction `letrec`. La construction :

$$(\texttt{letrec}\ (x\ e)\ e')$$

   est vue comme synonyme de

$$(\texttt{let}\ (x\ (\texttt{lambda}\ (y)\ y))\ (\texttt{setq}\ x\ e); e')$$

   Donner une règle de sémantique naturelle équivalente directe pour `letrec`. Prouvez l'équivalence dans le cas où e est de la forme $(\texttt{lambda}\ (z)\ e_1)$. Précisément, vous devrez prouver qu'un jugement $\rho \vdash (\texttt{letrec}\ (x\ e)\ e') \Rightarrow \rho', V$ est dérivable par votre règle si et seulement si $\rho \vdash (\texttt{let}\ (x\ (\texttt{lambda}\ (y)\ y))\ (\texttt{setq}\ x\ e); e') \Rightarrow \rho', V$ l'est.

4. Écrire la dérivation de sémantique naturelle la plus générale pour l'expression

$$\begin{array}{l} (\texttt{setq}\ f\ (\texttt{lambda}\ (x)\ (g\ x))); \\ (\texttt{let}\ (g\ (\texttt{lambda}\ (y)\ \texttt{deux})) \\ \quad (f\ \texttt{un})) \end{array}$$

   dans un environnement initial ρ qui à `un` associe la constante $1 \in \mathbb{Z}$ et à `deux` associe $2 \in \mathbb{Z}$.

5. Donner une définition intuitive de l'ensemble des variables libres d'une expression Lisp. Vous pourrez vous inspirer de la définition des variables libres d'une expression mini-Caml.

6. Pourquoi, contrairement à mini-Caml, la sémantique d'une expression Lisp ne dépend-elle pas que de ses variables libres ?

7. Sous quelles hypothèses sur l'environnement ρ la sémantique d'une expression e (à savoir l'ensemble des ρ', V tels que $\rho \vdash e \Rightarrow \rho', V$ est dérivable) ne dépend-elle que des variables libres de e ?

8. La question 4 illustre le fait que Lisp est un langage à *liaison dynamique* : la valeur des variables dépend du moment où l'on en est dans l'exécution du programme. À l'opposé, dans un langage à *portée statique* comme Caml, la valeur des variables ne dépend que de l'endroit dans le texte où elles sont définies.

   On souhaite définir un langage dont les expressions sont les expressions Lisp, mais qui soit à portée statique comme Caml. Noter qu'une différence importante par rapport à Caml est que l'on peut effectuer des effets de bord sur les variables. Donnons quelques exemples de comportements souhaités, en restant informel ; ce sera votre rôle de formaliser la suite. D'abord, l'exemple de la question 4 devrait maintenant retourner comme valeur $G(1)$, si $\rho(g)$ est la fonction G, et $\rho(\texttt{un}) = 1$. Ensuite, on souhaite que les valeurs des fonctions soient des clôtures, mais que les clôtures soient modifiables : après avoir effectué $(\texttt{setq}\ x\ 0); (\texttt{setq}\ (f\ (\texttt{lambda}\ (y)\ (\texttt{setq}\ x\ (+\ x\ 1)); x)))$ (après avoir étendu le langage de sorte que 0 dénote zéro, 1 un et $(+\ e\ e')$ l'addition de e et de e' évaluée de gauche à droite), on souhaite que le premier appel à f retourne 1, le deuxième retourne 2, etc.

   Donner une nouvelle sémantique pour Lisp qui tienne compte de ces besoins. Elle devra rester la plus simple et la plus élégante possible. Indication : vous pouvez imaginer une traduction de Lisp vers mini-Caml et déduire la nouvelle sémantique de Lisp à partir de celle de mini-Caml.