Traduction des langages

Sémantique opérationnelle et interprétation

Objectif : L'objectif de ce thème est d'apprendre à exprimer la sémantique d'exécution (sémantique dynamique) d'un langage en exploitant la notation de la déduction naturelle. Il s'agit d'une sémantique opérationnelle qui calcule la valeur de chaque expression.

Lors du dernier TP de programmation fonctionnelle, un évaluateur d'une sous-partie du langage a été écrit, il s'agit d'une partie de l'évaluateur que nous allons spécifier ici (et implanter en TP).

Le langage considéré est un sous-ensemble de OCaml.

1 Le langage mini-ML

1.1 Syntaxe du langage

La syntaxe du langage mini-ML que nous utiliserons comme support est définie par la grammaire suivante :

$$
\begin{array}{lcl}
\textit{Expr} & \rightarrow & \textit{Ident} \\
 & | & \textit{Const} \\
 & | & \textit{Expr Binaire Expr} \\
 & | & \textit{Unaire Expr} \\
 & | & \texttt{(}\ \textit{Expr}\ \texttt{)} \\
 & | & \texttt{if}\ \textit{Expr}\ \texttt{then}\ \textit{Expr}\ \texttt{else}\ \textit{Expr} \\
 & | & \texttt{let}\ \textit{Ident}\ \texttt{=}\ \textit{Expr}\ \texttt{in}\ \textit{Expr} \\
 & | & \texttt{fun}\ \textit{Ident}\ \texttt{->}\ \textit{Expr} \\
 & | & \texttt{(}\textit{Expr}\ \texttt{(}\textit{Expr}\texttt{))} \\
 & | & \texttt{let rec}\ \textit{Ident}\ \texttt{=}\ \textit{Expr}\ \texttt{in}\ \textit{Expr} \\
 & & \\
\textit{Const} & \rightarrow & \textit{entier} \mid \textit{booleen} \\
 & & \\
\textit{Unaire} & \rightarrow & \texttt{-} \mid \texttt{!} \\
 & & \\
\textit{Binaire} & \rightarrow & \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/} \mid \texttt{\%} \mid \texttt{\&} \mid \texttt{|} \\
 & | & \texttt{==} \mid \texttt{!=} \mid \texttt{<} \mid \texttt{<=} \mid \texttt{>} \mid \texttt{>=}
\end{array}
$$

1.2 Représentation des valeurs

Les valeurs possibles pour une expression en langage mini-ML sont décrites par la syntaxe suivante :

$$
\begin{array}{lcl}
\textit{Valeur} & \rightarrow & \textit{Const} \\
 & | & \bot
\end{array}
$$

Cette syntaxe pourra être étendue si nécessaire.

2 Sémantique opérationnelle

2.1 Quelle sémantique pour notre langage ?

Lors du dernier TP de programmation fonctionnelle une partie de l'interpréteur a été écrit (constantes, expressions binaires, définitions et utilisation d'identifiants).

La sémantique d'évaluation n'a cependant pas été définie clairement et vous avez peut-être des évaluations différentes pour une même expressions.

Par exemple, le code `let x = 1 in let x = 2 in x` renvoie :

— une erreur ? (double déclaration)

— 1 ? (la seconde déclaration est ignorée)

— 2 ? (la seconde déclaration masque la première)

Si nous voulons garder la sémantique d'Ocaml, l'évaluation doit renvoyer 2, mais nous pourrions envisager d'autres comportements. Il est donc important d'exprimer de façon formelle la sémantique de notre langage pour pouvoir avoir des garanties sur le code écrit.

D'autre part, cette implantation a mis en évidence le besoin d'un environnement (association identifiant - valeur) pour la gestion des identifiants.

2.2 Spécification formelle de la sémantique opérationnelle

Un jugement d'évaluation s'écrit sous la forme $\gamma \vdash e \Rightarrow v$, où :

— γ est un environnement (association $Ident$ / $Valeur$) ;

— e est une expression ($Expr$) ;

— v est une valeur ($Valeur$).

L'ensemble des axiomes et des règles de déduction correspondant à l'évaluation des expressions binaires, unaires, des identifiants et des constantes est présenté ci-dessous :

$$\gamma \vdash entier \Rightarrow entier \qquad \gamma \vdash booleen \Rightarrow booleen$$

$$\frac{x \in \gamma \quad \gamma(x) = v}{\gamma \vdash x \Rightarrow v} \qquad \frac{x \notin \gamma}{\gamma \vdash x \Rightarrow \bot_{undef}}$$

$$\frac{\gamma \vdash e_1 \Rightarrow v_1 \quad \gamma \vdash e_2 \Rightarrow v_2 \quad v_1 \times v_2 \in dom\, op \quad v = v_1\, op\, v_2}{\gamma \vdash e_1\, op\, e_2 \Rightarrow v}$$

$$\frac{\gamma \vdash e_1 \Rightarrow v_1 \quad \gamma \vdash e_2 \Rightarrow v_2 \quad v_1 \times v_2 \notin dom\, op}{\gamma \vdash e_1\, op\, e_2 \Rightarrow \bot_{type}}$$

$$\frac{\gamma \vdash e \Rightarrow v \quad v \in dom\, op \quad v' = op\, v}{\gamma \vdash op\, e \Rightarrow v'}$$

$$\frac{\gamma \vdash e \Rightarrow v \quad v \notin dom\, op}{\gamma \vdash op\, e \Rightarrow \bot_{type}}$$

3 Exercices

3.1 Ajout de la conditionnelle

$$Expr \quad \rightarrow \quad \texttt{if}\ Expr\ \texttt{then}\ Expr\ \texttt{else}\ Expr$$

Proposez des règles d'exécution pour la conditionnelle. Vous traiterez également les cas d'erreur.

3.2 Ajout de la définition locale

$$Expr \quad \rightarrow \quad \texttt{let}\ Ident\ \texttt{=}\ Expr\ \texttt{in}\ Expr$$

Proposez des règles d'exécution pour la définition locale. Vous traiterez également les cas d'erreur.

3.3 Ajout de la définition et l'appel de fonction

$$\begin{array}{rcl} Expr & \rightarrow & \texttt{fun}\ Ident\ \texttt{->}\ Expr \\ & | & \texttt{(}Expr\ \texttt{(}Expr\texttt{))} \end{array}$$

Proposez des règles d'exécution pour la définition et l'appel de fonction. Vous traiterez également les cas d'erreur.

3.4 Ajout de la définition récursive

$$Expr \quad \rightarrow \quad \texttt{let rec}\ Ident\ \texttt{=}\ Expr\ \texttt{in}\ Expr$$

Proposez des règles d'exécution pour la définition récursive. Vous ne traiterez pas les cas d'erreur.