Sémantique et traduction des langages
Interprétation d'un sous-ensemble de Caml : mini-ML

1 Grammaire du langage complet

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

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

2 Typage

Les types possibles pour une expression du langage mini-ML sont décrits par la syntaxe suivante :

$$
\begin{array}{lcl}
\tau & \to & \alpha \mid \texttt{int} \mid \texttt{bool} \mid \texttt{unit} \\
 & | & \tau \to \tau \\
 & | & (\tau)
\end{array}
$$

Un jugement de typage s'écrit sous la forme $\sigma \vdash e : \tau$.
Rappelons l'ensemble des axiomes et des règles de déduction correspondant au typage de mini-ML.

Accès à l'environnement

$$\frac{x \in \sigma \quad \sigma(x) = \tau}{\sigma \vdash x : \tau}$$

Opérateur binaire

$$\frac{\sigma \vdash e_1 : \tau_1 \quad \sigma \vdash e_2 : \tau_2 \quad \tau_1 \times \tau_2 = dom\, op \quad \tau = codom\, op}{\sigma \vdash e_1\ op\ e_2 : \tau}$$

Opérateur unaire

$$\frac{\sigma \vdash e : \tau \quad \tau = dom\, op \quad \tau' = codom\, op}{\sigma \vdash op\ e : \tau'}$$

Conditionnelle

$$\frac{\sigma \vdash e_1 : \texttt{bool} \quad \sigma \vdash e_2 : \tau \quad \sigma \vdash e_3 : \tau}{\sigma \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \tau}$$

Définition locale

$$\frac{\sigma \vdash e_1 : \tau_1 \quad \sigma :: \{x : \tau_1\} \vdash e_2 : \tau_2}{\sigma \vdash \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 : \tau_2}$$

Définition de fonction

$$\frac{\sigma :: \{x : \tau_1\} \vdash e : \tau_2}{\sigma \vdash \texttt{fun } x \texttt{ -> } e : \tau_1 \rightarrow \tau_2}$$

Appel de fonction

$$\frac{\sigma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \sigma \vdash e_2 : \tau_1}{\sigma \vdash \texttt{(} e_1 \texttt{) } e_2 : \tau_2}$$

Définition récursive

$$\frac{\sigma :: \{x : \tau_1\} \vdash e_1 : \tau_1 \quad \sigma :: \{x : \tau_1\} \vdash e_2 : \tau_2}{\sigma \vdash \texttt{letrec } x \texttt{ = } e_1 \texttt{ in } e_2 : \tau_2}$$

Création de référence

$$\frac{\sigma \vdash e : \tau}{\sigma \vdash \texttt{ref } e : @\tau}$$

Accès en lecture à une référence

$$\frac{\sigma \vdash e : @\tau}{\sigma \vdash \texttt{!} \ e : \tau}$$

Accès en écriture à une référence

$$\frac{\sigma \vdash e_1 : @\tau \qquad \sigma \vdash e_2 : \tau}{\sigma \vdash e_1 \texttt{ := } e_2 : \texttt{unit}}$$

Séquence

$$\frac{\sigma \vdash e_1 : \texttt{unit} \qquad \sigma \vdash e_2 : \tau}{\sigma \vdash e_1 \texttt{ ; } e_2 : \tau}$$

Gestion des erreurs

Il faut ajouter à ces règles, celles d'apparition et propagation des erreurs.