Typer

IFT-6172

November 15, 2023

1 Syntaxe

Voici une représentation possible de la syntaxe de Typer Light:

$$
\begin{array}{rccl}
\textit{(label)} & l & \in & \mathsf{Strings} \\
\textit{(level)} & n & \in & \mathbb{N} \\
\textit{(sort)} & s & ::= & \mathsf{Type} \\
\textit{(exp)} & e, \tau & ::= & c \mid s \mid x \mid e_1\ e_2 \mid \mathsf{lambda}(x : \tau) \to e \mid (x : \tau_1) \to \tau_2 \\
 & & \mid & \mathsf{let}\ d\ \mathsf{in}\ e \\
 & & \mid & \mathsf{typecons}\ \overrightarrow{cs} \\
 & & \mid & \mathsf{datacons}\ \tau\ l \\
 & & \mid & \mathsf{case}\ e \mid br_1 \mid ... \mid br_n \\
\textit{(decl)} & d & ::= & d_1; d_2 \mid x : \tau \mid x = e \\
\textit{(cstr)} & cs & ::= & (l\ \overrightarrow{(x : \tau)}) \\
\textit{(branch)} & br & ::= & x \Rightarrow e \mid l\ \vec{x} \Rightarrow e \\
\textit{(ctx)} & \Gamma & ::= & \bullet \mid \Gamma, x : \tau \\
\textit{(prog)} & p & ::= & d
\end{array}
$$

2 Static semantics

Typer Light est construit sur la base d'un *pure type system* (PTS). La première ligne de la syntaxe de *exp* ci-dessus est justement celle d'un PTS. Les règles de ce PTS sont les suivantes:

$$
\begin{array}{l}
\mathcal{S} = \{\mathsf{Type}\} \\
\mathcal{A} = \{\mathsf{Type} : \mathsf{Type}\} \\
\mathcal{R} = \{(\mathsf{Type}, \mathsf{Type}, \mathsf{Type})\}
\end{array}
$$

Les règles de typages d'un PTS sont les suivantes:

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{s : \tau \in \mathcal{A}}{\Gamma \vdash s : \tau} \qquad \frac{\Gamma \vdash e_1 : (x{:}\tau_1) \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2[e_2/x]}$$

$$\frac{\Gamma \vdash \tau_1 : s \qquad \Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash \mathsf{lambda}(x{:}\tau_1) \to e : (x{:}\tau_1) \to \tau_2} \qquad \frac{\Gamma \vdash e : \tau_1 \qquad \Gamma \vdash \tau_2 : s \qquad \tau_1 \simeq \tau_2}{\Gamma \vdash e : \tau_2}$$

$$\frac{\Gamma \vdash \tau_1 : s_1 \qquad \Gamma \vdash \tau_2 : s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash (x{:}\tau_1) \to \tau_2 : s_3}$$

Les règles de typage du let peuvent s'écrire comme suit, où $\Gamma_p; \Gamma_d \vdash d \Rightarrow \Gamma'_p; \Gamma'_d$ est un jugement auxiliaire qui vérifie les déclarations d, où Γ_p garde trace des variables déjà déclarées (mais pas forcément définies) et Γ_d garde trace des variables définies.

$$\frac{\Gamma_p \vdash \tau : s}{\Gamma_p; \Gamma_d \vdash x{:}\tau \Rightarrow \Gamma_p, x{:}\tau; \Gamma_d}\ (\textsc{Decl})$$

$$\frac{\Gamma_p \vdash e : \tau \qquad x \notin \Gamma_p}{\Gamma_p; \Gamma_d \vdash x = e \Rightarrow \Gamma_p, x{:}\tau; \Gamma_d, x{:}\tau}\ (\textsc{Def})$$

$$\frac{\Gamma_p \vdash e : \tau \qquad \Gamma_p(x) = \tau}{\Gamma_p; \Gamma_d \vdash x = e \Rightarrow \Gamma_p; \Gamma_d, x{:}\tau}\ (\textsc{DefRec})$$

$$\frac{\Gamma_p; \Gamma_d \vdash d_1 \Rightarrow \Gamma_{p1}; \Gamma_{d1} \qquad \Gamma_{p1}; \Gamma_{d1} \vdash d_2 \Rightarrow \Gamma_{p2}; \Gamma_{d2}}{\Gamma_p; \Gamma_d \vdash d_1; d_2 \Rightarrow \Gamma_{p2}; \Gamma_{d2}}$$

$$\frac{\Gamma, \Gamma \vdash d \Rightarrow \Gamma', \Gamma' \qquad \Gamma' \vdash e : \tau}{\Gamma \vdash \mathsf{let}\ d\ \mathsf{in}\ e : \tau}$$

Les règles de typage des types algébriques peuvent se décrire comme suit:

$$\frac{\forall i, j. \qquad cs_i = (l\ \overrightarrow{(x_i{:}\tau_i)}) \qquad \Gamma, x_{i1}{:}\tau_{i1}, .., x_{ij-i}{:}\tau_{ij-1} \vdash \tau_{ij} : \mathsf{Type}}{\Gamma \vdash \mathsf{typecons}(x\ \overrightarrow{(x{:}\tau)})\ \overrightarrow{cs} : \overrightarrow{(x{:}\tau)} \to \mathsf{Type}}$$

$$\frac{\tau \simeq \mathsf{typecons}(x\ \overrightarrow{(x{:}\tau)})\ \overrightarrow{cs} \qquad cs_l = (l\ \overrightarrow{(x_l{:}\tau_l)})}{\Gamma \vdash \mathsf{datacons}\ \tau\ l : \overrightarrow{(x{:}\tau)} \to \overrightarrow{(x_l{:}\tau_l)} \to \tau\ \vec{x}}$$

$$\frac{\begin{array}{c}\Gamma \vdash e : \tau\ \vec{e'} \qquad \tau \simeq \mathsf{typecons}(x\ \overrightarrow{(x{:}\tau)})\ \overrightarrow{cs} \qquad \sigma = \vec{e'}/\vec{x} \\ \forall i. \qquad br_i = l\ \vec{x_i} \Rightarrow e \qquad cs_l = (l\ \overrightarrow{(x_l{:}\tau_l)}) \qquad \Gamma, \overrightarrow{x_i{:}\tau_l[\sigma]} \vdash e : \tau'\end{array}}{\Gamma \vdash \mathsf{case}\ e \mid br_1 \mid ... \mid br_n : \tau'}$$

3 Dynamic semantics

La sémantique dynamique inclut les réductions primitives suivantes:

$$(\mathsf{lambda}(x:\tau) \to e)\ v \quad \rightsquigarrow \quad e[v/x] \ (\textsc{Beta})$$

$$\mathsf{let}\ x = v; d\ \mathsf{in}\ e \quad \rightsquigarrow \quad (\mathsf{let}\ d\ \mathsf{in}\ e)[v/x] \ (\textsc{PlainLet})$$

$$\frac{E = \mathsf{let}\ x : \tau; x = v; d\ \mathsf{in}\ \bullet \qquad v' = v[E[x]/x]}{E[e] \quad \rightsquigarrow \quad (\mathsf{let}\ d\ \mathsf{in}\ e)[v'/x]} \ (\textsc{RecursiveLet})$$

$$\frac{\tau = \mathsf{typecons}(y\ \overrightarrow{(y:\_)})\ \overrightarrow{cs} \qquad |\vec{v_1}| = |\overrightarrow{(y:\_)}|}{\mathsf{case}\ (\mathsf{datacons}\ \tau\ l)\ \vec{v_1}\ \vec{v_2} \mid ... \mid l\ \vec{x} \Rightarrow e \mid ... \quad \rightsquigarrow \quad e[\vec{v_2}/\vec{x}]} \ (\textsc{Dispatch})$$

$$\frac{v = (\mathsf{datacons}\ \tau\ l)\ \vec{v_1} \qquad l \text{ is not handled by any branch in } \overrightarrow{br}}{\mathsf{case}\ v \mid \overrightarrow{br} \mid x \Rightarrow e \quad \rightsquigarrow \quad e[v/x]} \ (\textsc{Default})$$

Auxquelles il faudrait ajouter les règles de congruences

La règle d'équivalence $e_1 \simeq e_2$ correspond à $\exists e_3.e_1 \rightsquigarrow^* e_3 \wedge e_2 \rightsquigarrow^* e_3$, sauf que les règles d'évaluation utilisent l'appel par valeur, alors que la règle d'équivalence ne veut pas se limiter à ça, donc elle doit utiliser une variante de $\rightsquigarrow$ où les v sont remplacées par des e.