

# Typer

IFT-6172

October 19, 2020

## 1 Syntaxe

Voici une représentation possible de la syntaxe de Typer:

(label)	$l$	$\in$	Strings
(level)	$n$	$\in$	$\mathbf{N}$
(sort)	$s$	$::=$	Type $n$
(exp)	$e, \tau$	$::=$	$c \mid s \mid x \mid e_1 e_2 \mid \text{lambda}(x:\tau) \rightarrow e \mid (x:\tau_1) \rightarrow \tau_2$   $\text{let } d \text{ in } e$   $\text{typecons}(x \overrightarrow{(x:\tau)}) \overrightarrow{cs}$   $\text{datacons } \tau l$   $\text{case } e \mid br_1 \mid \dots \mid br_n$
(decl)	$d$	$::=$	$d_1; d_2 \mid x : \tau \mid x = e$
(cstr)	$cs$	$::=$	$(l \overrightarrow{(x:\tau)})$
(branch)	$br$	$::=$	$x \Rightarrow e \mid l \vec{x} \Rightarrow e$
(ctx)	$\Gamma$	$::=$	$\bullet \mid \Gamma, x:\tau$
(prog)	$p$	$::=$	$d$

## 2 Static semantics

Typer 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:

$$\begin{aligned} \mathcal{S} &= \{\text{Type } n \mid n \in \mathbf{N}\}, \\ \mathcal{A} &= \{\text{Type } n_1 : \text{Type } n_2 \mid n_2 = n_1 + 1\}, \\ \mathcal{R} &= \{(\text{Type } n_1, \text{Type } n_2, \text{Type } n_3) \mid n_3 = \max(n_1, n_2)\} \end{aligned}$$

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

$$\begin{array}{c}
\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \quad \frac{s : \tau \in \mathcal{A}}{\Gamma \vdash s : \tau} \quad \frac{\Gamma \vdash e_1 : (x:\tau_1) \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2[e_2/x]} \\
\\
\frac{\Gamma \vdash \tau_1 : s \quad \Gamma, x:\tau_1 \vdash e : \tau_2}{\Gamma \vdash \text{lambda}(x:\tau_1) \rightarrow e : (x:\tau_1) \rightarrow \tau_2} \quad \frac{\Gamma \vdash e : \tau_1 \quad \Gamma \vdash \tau_2 : s \quad \tau_1 \simeq \tau_2}{\Gamma \vdash e : \tau_2} \\
\\
\frac{\Gamma \vdash \tau_1 : s_1 \quad \Gamma \vdash \tau_2 : s_2 \quad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash (x:\tau_1) \rightarrow \tau_2 : s_3}
\end{array}$$

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  $\Gamma_p$  garde trace des variables déjà déclarées (mais pas forcément définies) et  $\Gamma_d$  garde trace des variables définies.

$$\begin{array}{c}
\frac{\Gamma_p \vdash \tau : s}{\Gamma_p; \Gamma_d \vdash x:\tau \Rightarrow \Gamma_p, x:\tau; \Gamma_d} \text{ (DECL)} \\
\\
\frac{\Gamma_p \vdash e : \tau \quad x \notin \Gamma_p}{\Gamma_p; \Gamma_d \vdash x = e \Rightarrow \Gamma_p, x:\tau; \Gamma_d, x:\tau} \text{ (DEF)} \\
\\
\frac{\Gamma_p \vdash e : \tau \quad \Gamma_p(x) = \tau}{\Gamma_p; \Gamma_d \vdash x = e \Rightarrow \Gamma_p; \Gamma_d, x:\tau} \text{ (DEFREC)} \\
\\
\frac{\Gamma_p; \Gamma_d \vdash d_1 \Rightarrow \Gamma_{p1}; \Gamma_{d1} \quad \Gamma_{p1}; \Gamma_{d1} \vdash d_1 \Rightarrow \Gamma_{p2}; \Gamma_{d2}}{\Gamma_p; \Gamma_d \vdash d_1; d_2 \Rightarrow \Gamma_{p2}; \Gamma_{d2}} \\
\\
\frac{\Gamma, \Gamma \vdash d \Rightarrow \Gamma', \Gamma' \quad \Gamma' \vdash e : \tau}{\Gamma \vdash \text{let } d \text{ in } e : \tau}
\end{array}$$

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

$$\begin{array}{c}
\frac{\forall i, j. \quad cs_i = (l \overrightarrow{(x_i:\tau_i)}) \quad \Gamma, x_{i1}:\tau_{i1}, \dots, x_{ij-i}:\tau_{ij-i-1} \vdash \tau_{ij} : \text{Type } n_{ij}}{\Gamma \vdash \text{typecons}(x \overrightarrow{(x:\tau)}) \overrightarrow{cs} : (x:\tau) \rightarrow \text{Type} (\max n_{ij})} \\
\\
\frac{\tau \simeq \text{typecons}(x \overrightarrow{(x:\tau)}) \overrightarrow{cs} \quad cs_l = (l \overrightarrow{(x_l:\tau_l)})}{\Gamma \vdash \text{datacons } \tau \text{ l} : (x:\tau) \rightarrow \overrightarrow{(x_l:\tau_l)} \rightarrow \tau \vec{x}} \\
\\
\frac{\Gamma \vdash e : \tau \vec{e}' \quad \tau \simeq \text{typecons}(x \overrightarrow{(x:\tau)}) \overrightarrow{cs} \quad \sigma = \vec{e}'/\vec{x}}{\forall i. \quad br_i = l \vec{x}_i \Rightarrow e \quad cs_l = (l \overrightarrow{(x_l:\tau_l)}) \quad \Gamma, x_i:\tau_l[\sigma] \vdash e : \tau'} \\
\Gamma \vdash \text{case } e \mid br_1 \mid \dots \mid br_n : \tau'
\end{array}$$

### 3 Dynamic semantics

La sémantique dynamique inclut les réductions primitives suivantes:

$$(\text{lambda}(x:\tau) \rightarrow e) v \rightsquigarrow e[v/x] \text{ (BETA)}$$

$$\text{let } x = v; d \text{ in } e \rightsquigarrow (\text{let } d \text{ in } e)[v/x] \text{ (PLAINLET)}$$

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

$$\frac{\tau = \text{typecons}(y \overrightarrow{(y:\cdot)}) \overrightarrow{c^k} \quad |\vec{v}_1| = |\overrightarrow{(y:\cdot)}|}{\text{case } (\text{datacons } \tau l) \vec{v}_1 \vec{v}_2 \mid \dots \mid l \vec{x} \Rightarrow e \mid \dots \rightsquigarrow e[\vec{v}_2/\vec{x}]} \text{ (DISPATCH)}$$

$$\frac{v = (\text{datacons } \tau l) \vec{v}_1 \quad l \text{ is not handled by any branch in } \overrightarrow{br}}{\text{case } v \mid \overrightarrow{br} \mid x \Rightarrow e \rightsquigarrow e[v/x]} \text{ (DEFAULT)}$$

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  $e$ .