A Polymoprhic Typing Rules

Note: We will say a monomorphic type τ is an ***instance*** of a polymorphic type σ if there exists a monomorphic type τ', (a possibly empty) list of type variables $\alpha_1, \ldots, \alpha_n$, and a corresponding list of monomorphic types $\tau_1, \ldots, \tau_n$ such that $\sigma = \forall \alpha_1 \ldots \alpha_n. \tau'$ and $\tau = \tau'[\tau_1/\alpha_1; \ldots; \tau_n/\alpha_n]$, the type gotten by replacing each occurence of α_i in τ' by τ_i.

When using the rule below that require one type to be an instance of another, you should give the instantiation: $[\tau_1/\alpha_1; \ldots; \tau_n/\alpha_n]$.

A.1 Signatures:

Polymorphic constant signatures:

$\mathsf{sig}(\texttt{true}) = \texttt{bool}$	$\mathsf{sig}(\texttt{false}) = \texttt{bool}$
$\mathsf{sig}(n) = \texttt{int}$ $\quad n$ an integer constant	$\mathsf{sig}(f) = \texttt{float}$ $\quad f$ a floating point (real) constant
$\mathsf{sig}(s) = \texttt{string}$ $\quad s$ a string constant	$\mathsf{sig}(\texttt{[ ]}) = \forall \alpha. \alpha\texttt{list}$
$\mathsf{sig}(\texttt{( )}) = \texttt{unit}$	

Polymorphic Unary Primitive Operators:

$\mathsf{sig}(\texttt{fst}) = \forall \alpha\beta. (\alpha * \beta) \to \alpha$	$\mathsf{sig}(\texttt{snd}) = \forall \alpha\beta. (\alpha * \beta) \to \beta$
$\mathsf{sig}(\texttt{hd}) = \forall \alpha. \alpha\ \texttt{list} \to \alpha$	$\mathsf{sig}(\texttt{tl}) = \forall \alpha. \alpha\ \texttt{list} \to \alpha\ \texttt{list}$
$\mathsf{sig}(\sim) = \texttt{int} \to \texttt{int}$	$\mathsf{sig}(\texttt{print\_string}) = \texttt{string} \to \texttt{unit}$
$\mathsf{sig}(\texttt{not}) = \texttt{bool} \to \texttt{bool}$	

Polymorphic Binary Primitive Operators:

$\mathsf{sig}(\oplus) = \texttt{int} \to \texttt{int} \to \texttt{int}$ for $\oplus \in \{+, -, *, \texttt{mod}, /\}$	$\mathsf{sig}(\texttt{\^{}}) = \texttt{string} \to \texttt{string} \to \texttt{string}$
$\mathsf{sig}(\otimes) = \texttt{float} \to \texttt{float} \to \texttt{float}$ for $\otimes \in \{+., -., *., /., **\}$	$\mathsf{sig}((\_, \_)) = \forall \alpha\beta. \alpha \to \beta \to \alpha * \beta$
$\mathsf{sig}(\wr) = \texttt{bool} \to \texttt{bool} \to \texttt{bool}$ for $\wr \in \{\texttt{\|\|}, \texttt{\&\&}\}$	$\mathsf{sig}(::) = \forall \alpha. \alpha \to \alpha\ \texttt{list} \to \alpha\ \texttt{list}$
$\mathsf{sig}(\approx) = \forall \alpha. \alpha \to \alpha \to \texttt{bool}$ for $\approx \in \{\texttt{<}, \texttt{>}, \texttt{=}, \texttt{<=}, \texttt{>=}, \texttt{<>}\}$	

A.2 Rules:

Constants:

$$\frac{}{\Gamma \vdash c : \tau}\ \textsc{Const}$$ where c is a constant listed above, and τ is an instance of $\mathsf{sig}(c)$

Variables:

$$\frac{}{\Gamma \vdash x : \tau}\ \textsc{Var}$$ where x is a variable and τ is an instance of $\Gamma(x)$

Unary Primitive Operators:

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \oslash e : \tau_2}\ \textsc{Monop} \qquad \tau_1 \rightarrow \tau_2 \text{ an instance of } \mathsf{sig}(\oslash).$$

Binary Primitive Operators:

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 \odot e_2 : \tau_3}\ \textsc{Binop} \qquad \tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \text{ an instance of } \mathsf{sig}(\odot).$$

If_then_else rule:

$$\frac{\Gamma \vdash e_c : \texttt{bool} \quad \Gamma \vdash e_t : \tau \quad \Gamma \vdash e_e : \tau}{\Gamma \vdash \texttt{if}\ e_c\ \texttt{then}\ e_t\ \texttt{else}\ e_e : \tau}\ \textsc{If}$$

Application rule:

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}\ \textsc{App}$$

Function rule:

$$\frac{[x : \tau_1] + \Gamma \vdash e : \tau_2}{\Gamma \vdash \texttt{fun}\ x\ \texttt{->}\ e : \tau_1 \rightarrow \tau_2}\ \textsc{Fun}$$

Let rule:

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad [x : \mathsf{Gen}(\tau_1, \Gamma)] + \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \tau_2}\ \textsc{Let}$$

Let Rec rule:

$$\frac{[x : \tau_1] + \Gamma \vdash e_1 : \tau_1 \quad [x : \mathsf{Gen}(\tau_1, \Gamma)] + \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let rec}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \tau_2}\ \textsc{Rec}$$

$\mathsf{Gen}(\tau, \Gamma) = \forall \alpha_1 \ldots \alpha_n . \tau$ where $\alpha_1, \ldots, \alpha_n$ are the type variables that occur in τ but do not occur free in Γ.