Reinterpreate Types

1 INTRO

Here is a formalization of our type system.

2 CORE LANGUAGE

First, we'll define a small core language with basic integers, booleans, and functions.

$$
\begin{array}{rcll}
e & ::= & \mathbb{Z} \mid \mathbb{B} \mid x \mid \texttt{fun}\ x \texttt{->} e \mid e\ e & \textit{expressions} \\
x & ::= & \textit{(identifiers)} & \textit{variables} \\
v & ::= & \mathbb{Z} \mid \mathbb{B} \mid \texttt{fun}\ x \texttt{->} e \mid x & \textit{values} \\
\tau & ::= & \texttt{int} \mid \texttt{bool} \mid \tau\ \texttt{->}\ \tau & \textit{types}
\end{array}
$$

Fig. 1. Core language grammar

The typing rules of the system is defined as following:

Definition 2.1 (Typing rules).

(1) $\models e : \texttt{int}$ iff $e \Longrightarrow v, v \in \mathbb{Z}$.
(2) $\models e : \texttt{bool}$ iff $e \Longrightarrow v, v \in \mathbb{B}$.
(3) $\models e : \tau_1\ \texttt{->}\ \tau_2$ iff $\forall v$ such that $\models v : \tau_1, \models e\ v : \tau_2$.

3 LANGUAGE EXTENSIONS

Next, we will define a couple of languages extensions and their corresponding typing rules.

$$
\begin{array}{rcll}
e & ::= & \cdots \mid a & \textit{expressions} \\
v & ::= & \cdots \mid a & \textit{values} \\
\tau & ::= & \cdots \mid \alpha \mid \tau \cup \tau \mid \tau \cap \tau \mid \{\tau \mid e\} \mid (x : \tau)\ \texttt{->}\ \tau \mid \mu\alpha.\tau & \textit{types}
\end{array}
$$

Fig. 2. Extended language grammar

Definition 3.1 (More typing rules).

(1) $\models e : \alpha_i$ iff $e \Longrightarrow a_i$.
(2) $\models e : \tau_1 \cup \tau_2$ iff $\models e : \tau_1$ or $\models e : \tau_2$.
(3) $\models e : \tau_1 \cap \tau_2$ iff $\models e : \tau_1$ and $\models e : \tau_2$.
(4) $\models e : \{\tau \mid p\}$ iff $\models e : \tau$ and $p\ e \Longrightarrow \texttt{true}$.
(5) $\models e : (x : \tau_1)\ \texttt{->}\ \tau_2$ iff $\forall v$ such that $\models v : \tau_1, \models e\ v : \tau_2[v/x]$.
(6) $\models e : \mu\alpha.\tau$ iff $e : \tau[\mu\alpha.\tau/\alpha]$.

We will now extend the language with records.

$$
\begin{array}{rcll}
e & ::= & \cdots \mid \{\overline{\ell = e}\}^{\{\overline{\ell}\}} \mid e.\ell & \textit{expressions} \\
v & ::= & \cdots \mid \{\overline{\ell = v}\}^{\{\overline{\ell}\}} & \textit{values} \\
\tau & ::= & \cdots \mid \{\overline{\ell : \tau}\} & \textit{types}
\end{array}
$$

Fig. 3. Extended language grammar (with records)

Definition 3.2 (Record typing rules).

(1) $\models e : \{\ell_1 : \tau_1, \cdots, \ell_m : \tau_m\}$ iff $e \Longrightarrow \{\ell_1 = v_1, \cdots, \ell_m = v_m, \cdots, \ell_n = v_n\}^{\{\ell_1, \cdots, \ell_m\}}$ where $\models v_i : \tau_i$ for $i \in \{1, \cdots, m\}$, $n \geq m$

4 TYPE AS VALUES

In this section, we will demonstrate how to represent each type using a tuple of functions `generator` and `checker`.

Definition 4.1 (Semantic interpretation of types). We define the semantic interpretation of types as $[\![\tau]\!]$, where $[\![\tau]\!] = \langle \mathtt{generator}(\tau), \mathtt{checker}(\tau, e)\rangle$.

Definition 4.2 (Defining Generator in the core language).

(1) $\mathtt{generator(int)} : \mathtt{pick}\ n \in \mathbb{Z}$.
(2) $\mathtt{generator(bool)} : \mathtt{pick}\ b \in \mathbb{B}$.
(3) $\mathtt{generator}(\tau_1\ \texttt{->}\ \tau_2) : \mathtt{fun}\ x\ \texttt{->}\ \mathtt{generator}(\tau_2)$.

Definition 4.3 (Defining Checker in the core language).

(1) $\mathtt{checker(int}, e) : e \sim \mathtt{int}$.
(2) $\mathtt{checker(bool}, e) : e \sim \mathtt{bool}$.
(3) $\mathtt{checker}(\tau_1\ \texttt{->}\ \tau_2, e) : \mathtt{let\ arg\ =\ generator}(\tau_1)\ \mathtt{in\ checker}(\tau_2, \mathtt{(e\ arg)})$.

Definition 4.4 (Defining Generator in the extended language).

(1) $\mathtt{generator}(\alpha_i) : a_i$.
(2) $\mathtt{generator}(\tau_1 \cup \tau_2) : \mathtt{pick}\ b \in \mathbb{B}.\ \mathtt{if}\ b\ \mathtt{then\ generator}(\tau_1)\ \mathtt{else\ generator}(\tau_2)$.
(3) $\mathtt{generator}(\tau_1 \cap \tau_2)$ where τ_1, τ_2 are not arrow types : $\mathtt{pick}\ b \in \mathbb{B}$.
$\mathtt{if}\ b\ \mathtt{then\ let\ gend\ =\ generator}(\tau_1)\ \mathtt{in\ take(checker}(\tau_2, \mathtt{gend}), \mathtt{gend})\ \mathtt{else}$
$\mathtt{let\ gend\ =\ generator}(\tau_2)\ \mathtt{in\ take(checker}(\tau_1, \mathtt{gend}), \mathtt{gend})$.
(4) $\mathtt{generator}(\tau_1 \cap \tau_2)$ where $\tau_1 = \tau_{dom1}\ \texttt{->}\ \tau_{cod1}, \tau_2 = \tau_{dom2}\ \texttt{->}\ \tau_{cod2}$:
$\mathtt{fun}\ x\ \texttt{->}\ \mathtt{if\ checker}(\tau_{dom1}, x)\ \mathtt{then\ generator}(\tau_{cod1})\ \mathtt{else}$
$\mathtt{generator}(\tau_{cod2})$.
(5) $\mathtt{generator}(\tau_1 \cap \tau_2)$ where
$\tau_1 = \{\ell_1 : \tau_1, \cdots, \ell_m : \tau_m\}, \tau_2 = \{\ell_1 : \tau_1, \cdots, \ell_m : \tau_m, \cdots, \ell_n : \tau_n\}$:
$\mathtt{fun}\ x\ \texttt{->}\ \mathtt{if\ checker}(\tau_{dom1}, x)\ \mathtt{then\ generator}(\tau_{cod1})\ \mathtt{else}$
$\mathtt{generator}(\tau_{cod2})$.
(6) $\mathtt{generator}(\{\tau \mid p\}) : \mathtt{let\ choice\ =\ generator}(\tau)\ \mathtt{in\ take}(p, \mathit{choice})$.
(7) $\mathtt{generator}((x : \tau_1)\ \texttt{->}\ \tau_2) : \mathtt{let}\ \tau_2' = \mathtt{fun}\ x\ \texttt{->}\ \tau_2\ \mathtt{in\ fun}\ x'\ \texttt{->}\ \mathtt{generator}(\tau_2'\ x')$.
(8) $\mathtt{generator}(\mu\alpha.\tau) : \mathtt{generator}(\tau[\alpha/\mu\alpha.\tau])$.
(9) $\mathtt{generator}(\{\ell_1 : \tau_1, \cdots, \ell_n : \tau_n\})$:
$\mathtt{let}\ v_1 = \mathtt{generator}(\tau_1)\ \mathtt{in}\ \cdots\ \mathtt{let}\ v_n = \mathtt{generator}(\tau_n)\ \mathtt{in}\ \{\ell_1 = v_1, \cdots, \ell_n = v_n\}$.

Definition 4.5 (Defining Checker in the extended language).

(1) $\mathtt{checker}(\alpha_i, e) : e \sim a_i$.
(2) $\mathtt{checker}(\tau_1 \cup \tau_2, e) : \mathtt{checker}(\tau_1, e)\ \mathtt{or\ checker}(\tau_2, e)$.
(3) $\mathtt{checker}(\tau_1 \cap \tau_2, e) : \mathtt{checker}(\tau_1, e)\ \mathtt{and\ checker}(\tau_2, e)$.
(4) $\mathtt{checker}(\{\tau \mid p\}, e) : \mathtt{checker}(\tau, e)\ \mathtt{and\ eval}(e) = \mathtt{true}$.
(5) $\mathtt{checker}((x : \tau_1)\ \texttt{->}\ \tau_2, e) : \mathtt{let\ arg\ =\ generator}(\tau_1)\ \mathtt{in\ checker}(\tau_2[\mathtt{arg}/x], \mathtt{(e\ arg)})$.
(6) $\mathtt{checker}(\mu\alpha.\tau, e) : \mathtt{checker}(\tau[\mu\alpha.\tau/\alpha], e)$.
(7) $\mathtt{checker}(\{\ell_1 : \tau_1, \cdots, \ell_m : \tau_m\}, e) : \mathtt{eval}(e) = \{\ell_1 = v_1, \cdots, \ell_m = v_m, \cdots, \ell_n = v_n\}^{\{\ell_1, \cdots, \ell_m\}}$
$\mathtt{and\ checker}(\tau_1, v_1) \cdots\ \mathtt{and\ checker}(\tau_m, v_m)$.