

Reinterpret 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.

e	$::= \mathbb{Z} \mid \mathbb{B} \mid x \mid \text{fun } x \rightarrow e \mid e \ e$	<i>expressions</i>
x	$::= (\text{identifiers})$	<i>variables</i>
v	$::= \mathbb{Z} \mid \mathbb{B} \mid \text{fun } x \rightarrow e \mid x$	<i>values</i>
τ	$::= \text{int} \mid \text{bool} \mid \tau \rightarrow \tau$	<i>types</i>

Fig. 1. Core language grammar

The typing rules of the system is defined as following:

Definition 2.1 (Typing rules).

- (1) $\models e : \text{int}$ iff $e \implies v, v \in \mathbb{Z}$.
- (2) $\models e : \text{bool}$ iff $e \implies v, v \in \mathbb{B}$.
- (3) $\models e : \tau_1 \rightarrow \tau_2$ 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.

e	$::= \dots \mid a$	<i>expressions</i>
v	$::= \dots \mid a$	<i>values</i>
τ	$::= \dots \mid \alpha \mid \tau \cup \tau \mid \tau \cap \tau \mid \{\tau \mid e\} \mid (x : \tau) \rightarrow \tau \mid \mu \alpha. \tau$	<i>types</i>

Fig. 2. Extended language grammar

Definition 3.1 (More typing rules).

- (1) $\models e : \alpha$ iff $e \implies a$, where $\text{TYPEOF}(a) = \alpha$.
- (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 $p \ e \implies \text{true}$.
- (5) $\models e : (x : \tau_1) \rightarrow \tau_2$ iff $\forall v$ such that $\models v : \tau_1, \models e \ v : \tau_2[v/x]$.
- (6) $\models e : \mu \alpha. \tau$ iff ?.

We will now extend the language with records.

e	$::= \dots \mid \{\overline{\ell = e}\}^{\{\bar{\ell}\}} \mid e. \ell$	<i>expressions</i>
v	$::= \dots \mid \{\overline{\ell = v}\}^{\{\bar{\ell}\}}$	<i>values</i>
τ	$::= \dots \mid \{\ell : \tau\}$	<i>types</i>

Fig. 3. Extended language grammar (with records)

Definition 3.2 (Record typing rules).

- (1) $\models e : \{\ell_1 : \tau_1, \dots, \ell_m : \tau_m\}$ iff $e \implies \{\ell_1 = v_1, \dots, \ell_m = v_m, \dots, \ell_n = v_n\}^{\{\ell_1, \dots, \ell_p\}}$ where $\models v_i : \tau_i$ for $i \in \{1, \dots, m\}, n \geq p \geq m$.

$e ::= \mathbb{Z} \mid \mathbb{B} \mid x \mid \text{fun } x \rightarrow e \mid e \ e \mid e \odot e \mid a \mid \{\overline{\ell} = e\}^{\{\bar{\ell}\}} \mid e.\ell$	<i>expressions</i>
$\mid \text{if } e \text{ then } e \text{ else } e \mid \text{pick}_i \mid \text{pick}_b \mid \text{mzero} \mid \text{ERROR}$	
$\mid \text{let } x = e \text{ in } e \mid \text{let } f \ x = e \text{ in } e \mid e \sim p$	
$x ::= (\text{identifiers})$	<i>variables</i>
$p ::= \text{int} \mid \text{bool} \mid \text{fun} \mid \text{any} \mid a \mid \{\bar{\ell}\}$	<i>patterns</i>
$v ::= \mathbb{Z} \mid \mathbb{B} \mid \text{fun } x \rightarrow e \mid x \mid a \mid \{\overline{\ell} = v\}^{\{\bar{\ell}\}}$	<i>values</i>
$\tau ::= \text{int} \mid \text{bool} \mid \tau \rightarrow \tau \mid \alpha \mid \tau \cup \tau \mid \tau \cap \tau \mid \{\tau \mid e\} \mid (x : \tau) \rightarrow \tau \mid \mu\alpha.\tau \mid \{\bar{\ell} : \tau\}$	<i>types</i>

Fig. 4. Complete language grammar

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 $\llbracket \tau \rrbracket = \{\text{gen} = \text{generator}(\tau), \text{check} = \text{fun } e \rightarrow \text{checker}(\tau, e)\}$.

Definition 4.2 (Defining Generator in the core language).

- (1) $\text{generator}(\text{int}) : \text{pick}_i$.
- (2) $\text{generator}(\text{bool}) : \text{pick}_b$.
- (3) $\text{generator}(\tau_1 \rightarrow \tau_2) : \text{fun } x \rightarrow \text{generator}(\tau_2)$.

Definition 4.3 (Defining Checker in the core language).

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

Definition 4.4 (Defining Generator in the extended language).

- (1) $\text{generator}(\alpha_i) : a_i$.
- (2) $\text{generator}(\tau_1 \cup \tau_2) : \text{pick}_b. \text{if } b \text{ then } \text{generator}(\tau_1) \text{ else } \text{generator}(\tau_2)$.
- (3) $\text{generator}(\tau_1 \cap \tau_2)$ where τ_1, τ_2 are not arrow types or record types : $\text{pick } b \in \mathbb{B}$.
 $\text{if } b \text{ then}$
 $\quad \text{let } \text{gend} = \text{generator}(\tau_1) \text{ in}$
 $\quad \text{if } \text{checker}(\tau_2, \text{gend}) \text{ then } \text{gend} \text{ else } \text{mzero}$
 else
 $\quad \text{let } \text{gend} = \text{generator}(\tau_2) \text{ in}$
 $\quad \text{if } \text{checker}(\tau_1, \text{gend}) \text{ then } \text{gend} \text{ else } \text{mzero}$
- (4) $\text{generator}(\tau_1 \cap \tau_2)$ where $\tau_1 = \tau_{\text{dom1}} \rightarrow \tau_{\text{cod1}}, \tau_2 = \tau_{\text{dom2}} \rightarrow \tau_{\text{cod2}} :$
 $\text{fun } x \rightarrow$
 $\quad \text{if } \text{checker}(\tau_{\text{dom1}}, x) \text{ then } \text{generator}(\tau_{\text{cod1}}) \text{ else } \text{generator}(\tau_{\text{cod2}})$.
- (5) $\text{generator}(\tau_1 \cap \tau_2)$ where
 $\tau_1 = \{\ell_1 : \tau'_1, \dots, \ell_n : \tau'_n, \dots, \ell_{11} : \tau'_{11}, \dots, \ell_{1m} : \tau'_{1m}\},$
 $\tau_2 = \{\ell_1 : \tau''_1, \dots, \ell_n : \tau''_n, \dots, \ell_{21} : \tau''_{21}, \dots, \ell_{2n} : \tau''_{2n}\} :$
 $\{\ell_1 = \text{generator}(\tau'_1 \cap \tau''_1), \dots, \ell_n = \text{generator}(\tau'_n \cap \tau''_n), \dots, \ell_{11} = \tau_{11}, \dots, \ell_{2n} = \tau'_{2n}\}.$
- (6) $\text{generator}(\{\tau \mid p\}) :$
 $\text{let } \text{gend} = \text{generator}(\tau) \text{ in if } (p \ \text{gend}) \text{ then } \text{gend} \text{ else } \text{mzero}.$

- (7) $\text{generator}((x : \tau_1) \rightarrow \tau_2) :$
 $\text{fun } x' \rightarrow \text{if checker}(\tau_1, x') \text{ then generator}(\tau_2[x'/x]) \text{ else ERROR.}$
- (8) $\text{generator}(\mu\alpha.\tau) : \text{generator}(\tau[\alpha/\mu\alpha.\tau]).$
- (9) $\text{generator}(\{\ell_1 : \tau_1, \dots, \ell_n : \tau_n\}) :$
 $\text{let } v_1 = \text{generator}(\tau_1) \text{ in } \dots \text{let } v_n = \text{generator}(\tau_n) \text{ in } \{\ell_1 = v_1, \dots, \ell_n = v_n\}.$

Definition 4.5 (Defining Checker in the extended language).

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

5 SELECTIVE TYPECHECKING

We allow users to declare types in their program selectively. If an expression doesn't have a type declaration, we assume that the user does not wish for us to check its type. In other words, we will only be checking explicitly declared types in the user program.

$e ::= \dots \mid \text{let } f \ (x : \tau) : \tau = e \text{ in } e \mid \text{let } (x : \tau) = e \text{ in } e \quad \text{expressions}$

Fig. 5. Updated language grammar