

A Simple Type System

Types play a pivotal rôle in the design of modern programming languages. We here present a simple type system ST and establish its type soundness. The syntax of ST is given as follows:

$$\begin{array}{ll}
 \text{booleans} & b ::= \text{true} \mid \text{false} \\
 \text{integers} & i ::= 0 \mid -1 \mid 1 \mid -2 \mid 2 \mid \dots \\
 \text{terms} & t ::= b \mid i \mid x \mid c(t_1, \dots, t_n) \mid \langle t_1, t_2 \rangle \mid \mathbf{fst}(t) \mid \mathbf{snd}(t) \mid \mathbf{lam} \ x.t \mid \mathbf{app}(t_1, t_2) \mid \\
 & \mathbf{if}(t_1, t_2, t_3) \mid \mathbf{fix} \ f(x).t \\
 \text{values} & v ::= b \mid i \mid x \mid \langle v_1, v_2 \rangle \mid \mathbf{lam} \ x.t \\
 \text{types} & T ::= \mathbf{bool} \mid \mathbf{int} \mid T_1 * T_2 \mid T_1 \rightarrow T_2 \\
 \text{contexts} & \Gamma ::= \emptyset \mid \Gamma, x : T
 \end{array}$$

We use b and i for booleans and integers, respectively, and x for variables. There are terms and types in ST , and we are to present rules for assigning types to terms. We say that a term t is of type T if T can be assigned to t according to such type assignment rules. We use c for a builtin constant function such as addition (+) and subtraction (-), and $c(t_1, \dots, t_n)$ for the application of a constant function c to n arguments t_1, \dots, t_n . We use $\langle t_1, t_2 \rangle$ for forming a pair and $\mathbf{fst}(t)$ and $\mathbf{snd}(t)$ for the first and second projections. In addition, we use $\mathbf{lam} \ x.t$ for lambda-abstraction and $\mathbf{app}(t_1, t_2)$ for function application.

We use v for values, which are a special form of terms; both booleans and integers are values; a pair of values is a value; a lambda-abstract is also a value.

The types \mathbf{bool} and \mathbf{int} are for booleans and integers, respectively. Given types T_1 and T_2 , we can form a type $T_1 * T_2$ for pairs whose first and second components are of types T_1 and T_2 , respectively; also we can form a type $T_1 \rightarrow T_2$ for functions that returns a value of type T_2 when applied to an argument of type T_1 .

We use $\Gamma \vdash t : T$ for a typing judgment meaning that the term t can be assigned the type T under the context Γ . The rule for deriving typing judgments are give in Figure 1.

Lemma 1 (Canonical Forms) *Assume that $\emptyset \vdash v : T$ is derivable.*

1. If $T = \mathbf{bool}$, then v is a boolean value b .
2. If $T = \mathbf{int}$, then v is an integer value b .
3. If $T = T_1 * T_2$, then v is of the form $\langle v_1, v_2 \rangle$.
4. If $T = T_1 \rightarrow T_2$, then v is of the form $\mathbf{lam} \ x.t$.

Lemma 2 (Substitution) *Assume that $\Gamma \vdash t_1 : T_1$ and $\Gamma, x : T_1 \vdash t_2 : T_2$ are derivable. Then $\Gamma \vdash t_2[x \mapsto t_1] : T_2$ is also derivable.*

We write $t_1 \rightarrow t_2$ to mean that t_1 reduces to t_2 in one step, and the reduction rules are given in Figure 2.

$$\begin{array}{c}
\overline{\Gamma \vdash b : \mathbf{bool}} \quad (\mathbf{bool}) \\
\overline{\Gamma \vdash i : \mathbf{int}} \quad (\mathbf{int}) \\
\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad (\mathbf{var}) \\
\frac{\vdash c(T_1, \dots, T_n) : T \quad \Gamma \vdash t_1 : T_1 \quad \dots \quad \Gamma \vdash t_n : T_n}{\Gamma \vdash c(t_1, \dots, t_n) : T} \quad (\mathbf{const}) \\
\frac{\Gamma \vdash t_1 : T_1 \quad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \langle t_1, t_2 \rangle : T_1 * T_2} \quad (\mathbf{tup}) \\
\frac{\Gamma \vdash t : T_1 * T_2}{\Gamma \vdash \mathbf{fst}(t) : T_1} \quad (\mathbf{fst}) \\
\frac{\Gamma \vdash t : T_1 * T_2}{\Gamma \vdash \mathbf{snd}(t) : T_2} \quad (\mathbf{snd}) \\
\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \mathbf{lam} \ x.t : T_1 \rightarrow T_2} \quad (\mathbf{lam}) \\
\frac{\Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1}{\Gamma \vdash \mathbf{app}(t_1, t_2) : T_2} \quad (\mathbf{app}) \\
\frac{\Gamma \vdash t_1 : \mathbf{bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \mathbf{if}(t_1, t_2, t_3) : T} \quad (\mathbf{if}) \\
\frac{\Gamma, f : T_1 \rightarrow T_2, x : T_1 \vdash t : T_2}{\Gamma \vdash \mathbf{fix} \ f(x).t : T_1 \rightarrow T_2} \quad (\mathbf{fix})
\end{array}$$

Figure 1: The typing rules

$$\begin{array}{c}
\frac{t_i \rightarrow t'_i}{c(v_1, \dots, v_{i-1}, t_i, t_{i+1}, \dots, t_n) \rightarrow c(v_1, \dots, v_{i-1}, t'_i, t_{i+1}, \dots, t_n)} \\
\frac{c(v_1, \dots, v_n) = v}{c(v_1, \dots, v_n) \rightarrow v} \\
\frac{t_1 \rightarrow t'_1}{\langle t_1, t_2 \rangle \rightarrow \langle t'_1, t_2 \rangle} \\
\frac{t_2 \rightarrow t'_2}{\langle v_1, t_2 \rangle \rightarrow \langle v_1, t'_2 \rangle} \\
\frac{\mathbf{fst}(\langle v_1, v_2 \rangle) \rightarrow v_1}{\mathbf{fst}(\langle v_1, v_2 \rangle) \rightarrow v_1} \\
\frac{\mathbf{snd}(\langle v_1, v_2 \rangle) \rightarrow v_2}{\mathbf{snd}(\langle v_1, v_2 \rangle) \rightarrow v_2} \\
\frac{t_1 \rightarrow t'_1}{\mathbf{app}(t_1, t_2) \rightarrow \mathbf{app}(t'_1, t_2)} \\
\frac{t_2 \rightarrow t'_2}{\mathbf{app}(v_1, t_2) \rightarrow \mathbf{app}(v_1, t'_2)} \\
\mathbf{app}(\mathbf{lam } x.t, v) \rightarrow t[x \mapsto v] \\
\frac{t_1 \rightarrow t'_1}{\mathbf{if}(t_1, t_2, t_3) \rightarrow \mathbf{if}(t'_1, t_2, t_3)} \\
\frac{\mathbf{if}(\mathbf{true}, t_1, t_2) \rightarrow t_1}{\mathbf{if}(\mathbf{false}, t_1, t_2) \rightarrow t_2} \\
\mathbf{fix } f(x).t \rightarrow \mathbf{lam } x.t[f \mapsto \mathbf{fix } f(x).t]
\end{array}$$

Figure 2: The reduction rules

Theorem 1 (Subject Reduction) *Assume that $\emptyset \vdash t : T$ is derivable and $t \rightarrow t'$ holds. Then $\emptyset \vdash t \rightarrow t'$ is also derivable.*

Theorem 2 (Progress) *Assume that $\emptyset \vdash t : T$ is derivable. Then t is either a value, or $t \rightarrow t'$ for some term t' .*

We now present a slightly different means to assign dynamic semantics to terms in ST. We first introduce the notion of evaluation contexts:

$$\text{evaluation contexts } E ::= [] \mid c(v_1, \dots, v_{i-1}, E, t_{i+1}, \dots, t_n) \mid \langle E, t \rangle \mid \langle v, E \rangle \mid \mathbf{fst}(E) \mid \mathbf{snd}(E) \mid \mathbf{app}(E, t) \mid \mathbf{app}(v, E) \mid \mathbf{if}(E, t_2, t_3)$$

We then introduce the notion of redexes as follows:

Definition 3 *We define redexes and their reductions as follows.*

1. $c(v_1, \dots, v_n)$ is a redex if c is a built-in function and $c(v_1, \dots, v_n)$ is defined to equal v , and the reduction of $c(v_1, \dots, v_n)$ is v .
2. $\mathbf{app}(\mathbf{lam } x.t, v)$ is a redex, and its reduction is $t[x \mapsto v]$.
3. $\mathbf{fst}(\langle v_1, v_2 \rangle)$ is a redex, and its reduction is v_1 .
4. $\mathbf{snd}(\langle v_1, v_2 \rangle)$ is a redex, and its reduction is v_2 .
5. $\mathbf{if}(\mathbf{true}, t_1, t_2)$ is a redex, and its reduction is t_1 .
6. $\mathbf{if}(\mathbf{false}, t_1, t_2)$ is a redex, and its reduction is t_2 .
7. $\mathbf{fix } f(x).t$ is a redex and its reduction is $\mathbf{lam } x.t[f \mapsto \mathbf{fix } f(x).t]$.

Given $t_1 = E[t]$ and $t_2 = E[t']$ for some redex t and its reduction t' , we write $t_1 \rightarrow t_2$ and say that t_1 reduces to t_2 in one step. Let \rightarrow^* be the reflexive and transitive closure of \rightarrow .

We now make some additional adjustments in order to support the language constructs **callcc** and **throw**.

$$\text{terms } t ::= \dots \mid \mathbf{callcc}(\mathbf{lam } x.t) \mid \mathbf{throw}(t_1, t_2) \mid *E$$

$$\text{evaluation contexts } E ::= \dots \mid \mathbf{throw}(E, t) \mid \mathbf{throw}(v, E)$$

The new forms of evaluation rules are given as follows:

$$E[\mathbf{callcc}(\mathbf{lam } x.t)] \rightarrow E[t[x \mapsto *E]] \qquad E[\mathbf{throw}(*E', v)] \rightarrow E'[v]$$