Here we extend our little language to have an infinite set of types with some interesting structure, including function and record types, and assignable variables with aliasing.

1 Syntax

Types

\[ \tau ::= \text{int} \mid \text{bool} \mid \text{string} \mid \text{var}(\tau) \mid \text{prod}(\ldots x_i : \tau_i \ldots) \mid \text{fun}(\tau_1)\tau_2 \]

Addresses

\[ a^\tau \in A \quad a^\tau = \langle i, \tau \rangle \]

This represents a typed address constant – \( a^\tau \) identifies (“is the address of”) a value of type \( \tau \) in the store. We assume there are infinitely many address constants of each type, and they are ordered (e.g. numerically).

Expressions

\[ e ::= n \mid t \mid s \mid a^\tau \mid \psi e_1 \mid e_1 \theta e_1 \mid e_1; e_2 \mid \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \]
| (while $e_1$ do $e_2$) |
| newvar($e_1$) | $e_1$ $\leftarrow$ $e_2$ | $e_1$ $\uparrow$ |
| (let $\ldots$ $x_i$ $\sim$ $e_i$ $\ldots$ in $e_0$) | $x$ |
| ($\ldots$ $x_i$ $\sim$ $e_i$ $\ldots$ ) | $e.x$ |
| (lambda $x : \tau$ dot $e$) | $e_1(e_2)$ |

2 Typing Rules

2.1 Preliminaries

The typing rules follow the same general structure as the ones in Notes 8. That is, a type assignment $\pi$ is a finite set

$$\pi = \{ \ldots x_i \sim \tau_i \ldots \}$$

of assignments of types to names. The definitions of

$$\text{dom}(\pi) \quad \pi|_S \quad (\pi_1 \oplus \pi_2)$$

all appear in Notes 8. Judgements take the form

$$\pi \vdash e : \tau$$

and the well-typed program expressions are those $e$ such that

$$\{ \} \vdash e : \tau$$

is derivable for some $\tau$.

2.2 The Rules

Constants

$$\pi \vdash n : \text{int} \quad \text{(T9.1)}$$

$$\pi \vdash t : \text{bool} \quad \text{(T9.2)}$$
\( \pi \vdash s : \text{string} \)  \hspace{1cm} (T9.3)

\( \pi \vdash (a^\tau) : \text{var}(\tau) \)  \hspace{1cm} (T9.4)

**Operators**

\[
\frac{\pi \vdash e_1 : \tau_1 \quad \pi \vdash e_2 : \tau_2}{\pi \vdash (e_1 \theta e_2) : \tau} \quad \theta \text{ is } \tau_1 \times \tau_2 \to \tau
\]  \hspace{1cm} (T9.6)

\[
\frac{\pi \vdash e_1 : \tau_1 \quad \pi \vdash e_2 : \tau_2}{\pi \vdash (\psi e_1) : \tau} \quad \psi \text{ is } \tau_1 \to \tau
\]  \hspace{1cm} (T9.5)

**Control Structures**

\[
\frac{\pi \vdash e_1 : \tau_1 \quad \pi \vdash e_2 : \tau_2}{\pi \vdash (e_1 ; e_2) : \tau_2}
\]  \hspace{1cm} (T9.7)

\[
\frac{\pi \vdash e_1 : \text{bool} \quad \pi \vdash e_2 : \tau \quad \pi \vdash e_3 : \tau}{\pi \vdash (\text{if } e_1 \text{ then } e_2 \text{ else } e_3) : \tau}
\]  \hspace{1cm} (T9.8)

\[
\frac{\pi \vdash e_1 : \text{bool} \quad \pi \vdash e_2 : \tau}{\pi \vdash (\text{while } e_1 \text{ do } e_2) : \text{bool}}
\]  \hspace{1cm} (T9.9)

**Assignable Variables**

\[
\frac{\pi \vdash e_1 : \tau}{\pi \vdash \text{newvar}(e_1) : \text{var}(\tau)}
\]  \hspace{1cm} (T9.10)

\[
\frac{\pi \vdash e_1 : \text{var}(\tau) \quad \pi \vdash e_2 : \tau}{\pi \vdash (e_1 \leftarrow e_2) : \tau}
\]  \hspace{1cm} (T9.11)

\[
\frac{\pi \vdash e_1 : \text{var}(\tau)}{\pi \vdash (e_1 \uparrow) : \tau}
\]  \hspace{1cm} (T9.12)
Let Bindings

\[
\cdots \quad \pi \vdash e_i : \tau_i \quad \cdots \\
(\pi \oplus \{\ldots x_i : \tau_i \ldots\}) \vdash e_0 : \tau \\
(\pi \vdash \textbf{let} \ldots . \; x_i \sim e_i \ldots \; \textbf{in} \; e_0) : \tau
\]  

(T9.13)

\[
\pi \vdash x : \tau \quad \text{where } x : \tau \in \pi
\]  

(T9.14)

Products

\[
\cdots \quad \pi \vdash e_i : \tau_i \quad \cdots \\
\pi \vdash \langle \ldots . \; x_i \sim e_i \ldots \rangle : \textbf{prod}(\ldots . \; x_i : \tau_i \ldots)
\]  

(T9.15)

\[
\pi \vdash e : \textbf{prod}(\ldots . \; x_i : \tau_i \ldots) \\
\pi \vdash e.x_i : \tau_i
\]  

(T9.16)

Functions

\[
(\pi \oplus \{x : \tau'\}) \vdash e : \tau \\
\pi \vdash (\lambda \text{\textbf{lambda}} \; x : \tau' \; \textbf{dot} \; e) : \textbf{fun}(\tau')\tau
\]  

(T9.17)

\[
\pi \vdash e_1 : \textbf{fun}(\tau_2)\tau \\
\pi \vdash e_2 : \tau_2 \\
\pi \vdash e_1(e_2) : \tau
\]  

(T9.18)

2.3 Properties

The following are easily provable by induction on derivations:

**Prop** (unique typing) For any \( \pi, e, \tau, \tau' \)

\[(\pi \vdash e : \tau) \land (\pi \vdash e : \tau') \Rightarrow (\tau \equiv \tau')\]

That is, in any type environment an expression can be typed only one way. \( \Box \)

**Prop** (support) For any \( \pi, e, \tau \)

\[(\pi \vdash e : \tau) \Rightarrow ((\pi|\text{\textbf{fv}}(e)) \vdash e : \tau)\]

That is, the type of an expression depends only on the types of its free variables in the type assignment. \( \Box \)