Simply-Typed FUN

1 SimpleFUN Syntax

$$x \in \mathit{Variable} \qquad n \in \mathbb{N} \qquad b \in \mathit{Bool} \qquad \mathit{name}, \mathit{cons}, \mathit{fld} \in \mathit{Label}$$

$$\begin{aligned}
\mathit{prog} \in \mathit{Program} &::= \overrightarrow{\mathit{typedef}_i}\ e \\
\mathit{typedef} \in \mathit{TypeDef} &::= \textbf{type}\ \mathit{name} = \overrightarrow{\mathit{cons}_i : \tau_i} \\
e \in \mathit{Exp} &::= x \mid n \mid b \mid \textbf{nil} \mid (\overrightarrow{x_i : \tau_i}) \Rightarrow e \mid e_f(\vec{e_i}) \\
&\quad \mid \textbf{if}\ e_1\ e_2\ e_3 \mid \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \mid \textbf{rec}\ x : \tau = e_1\ \textbf{in}\ e_2 \\
&\quad \mid (\!|\overrightarrow{\mathit{fld}_i : e_i}|\!) \mid e.\mathit{fld} \mid \mathit{name}!\mathit{cons}\ e \mid \textbf{case}\ e\ \textbf{of}\ \overrightarrow{\mathit{cons}_i\ x_i \Rightarrow e_i}
\end{aligned}$$

We generalize from pairs to records and from unions to variants; we use nominal typing for variants to enable recursive datatypes. A program is a sequence of type definitions followed by an expression to evaluate. A type definition gives (1) a name for the user-defined type; and (2) a sequence of constructors that can be used to create something of that user-defined type, each given as the constructor name paired with the type of value that constructor takes. An expression is a variable, number, boolean, or **nil**; an anonymous function; a function call; a conditional; a variable declaration using **let**; a recursive variable definition using **rec**; a record consisting of a sequence of field names and their associated expressions; a record field access; a constructor call to create a variant value; or a pattern-matching expression on a variant value. Notice that there are no arithmetic, relational, or logical operators given in the syntax; for simplicity we'll assume that they are all given as builtin functions with the appropriate types: $\{+, -, \times, \div\} : (\textbf{num}, \textbf{num}) \to \textbf{num}$; $\{<, =\} : (\textbf{num}, \textbf{num}) \to \textbf{bool}$; $\{\wedge, \vee\} : (\textbf{bool}, \textbf{bool}) \to \textbf{bool}$; and $\neg : \textbf{bool} \to \textbf{bool}$.

2 SimpleFUN Type System

$$\tau \in \mathit{Type} = \textbf{num} \mid \textbf{bool} \mid \textbf{unit} \mid (\vec{\tau_i}) \to \tau_r \mid (\!|\overrightarrow{\mathit{fld}_i : \tau_i}|\!) \mid \mathit{name}$$

A type is **num** for numbers; **bool** for booleans; **unit** for the single unit value **nil**; an arrow type for a function with an arbitrary number of parameters; a record type with the set of field names and their associated types; or a user-defined variant type name.

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ \textsc{var} \qquad \frac{}{\Gamma \vdash n : \textbf{num}}\ n\text{I} \qquad \frac{}{\Gamma \vdash b : \textbf{bool}}\ b\text{I} \qquad \frac{}{\Gamma \vdash \textbf{nil} : \textbf{unit}}\ \textbf{nil}\text{I}$$

$$\frac{\Gamma, \overrightarrow{x_i : \tau_i} \vdash e : \tau_r}{\Gamma \vdash (\overrightarrow{x_i : \tau_i}) \Rightarrow e : (\vec{\tau_i}) \to \tau_r} \to\text{I} \qquad \frac{\Gamma \vdash e_f : (\vec{\tau_i}) \to \tau_r \quad \Gamma \vdash \overrightarrow{e_i : \tau_i}}{\Gamma \vdash e_f(\vec{e_i}) : \tau_r} \to\text{E} \qquad \frac{\Gamma \vdash e_1 : \textbf{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \textbf{if}\ e_1\ e_2\ e_3 : \tau}\ \textsc{if}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : \tau_2}\ \textsc{let} \qquad \frac{\Gamma, x : \tau_1 \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \textbf{rec}\ x : \tau_1 = e_1\ \textbf{in}\ e_2 : \tau_2}\ \textsc{rec}$$

$$\frac{\Gamma \vdash \overrightarrow{e_i : \tau_i}}{\Gamma \vdash (\!|\overrightarrow{\mathit{fld}_i : e_i}|\!) : (\!|\overrightarrow{\mathit{fld}_i : \tau_i}|\!)}\ \textsc{rcd}I \qquad \frac{\Gamma \vdash e : (\!|\overrightarrow{\mathit{fld}_i : \tau_i}|\!) \quad j \in \vec{i}}{\Gamma \vdash e.\mathit{fld}_j : \tau_j}\ \textsc{rcd}E$$

$$\frac{\textbf{type}\ \mathit{name} = \overrightarrow{\mathit{cons}_i : \tau_i} \in \mathit{TypeDef} \quad j \in \vec{i} \quad \Gamma \vdash e : \tau_j}{\Gamma \vdash \mathit{name}!\mathit{cons}_j\ e : \mathit{name}}\ \textsc{td}I \qquad \frac{\Gamma \vdash e : \mathit{name} \quad \textbf{type}\ \mathit{name} = \overrightarrow{\mathit{cons}_i : \tau_i} \in \mathit{TypeDef} \quad \overrightarrow{\Gamma, x_i : \tau_i \vdash e_i : \tau}}{\Gamma \vdash \textbf{case}\ e\ \textbf{of}\ \overrightarrow{\mathit{cons}_i\ x_i \Rightarrow e_i} : \tau}\ \textsc{td}E$$