Polymorphically-Typed FUN

1 PolyFUN Syntax

\[ x \in \text{Variable} \quad n \in \mathbb{N} \quad b \in \text{Bool} \quad \text{name, cons, fld} \in \text{Label} \]

\[
\text{prog} \in \text{Program} ::= \text{typedef}_1 \ldots \text{typedef}_n \ e \\
\text{typedef} \in \text{TypeDef} ::= \text{type name}[T_1 \ldots T_k] = \text{cons}_1 : \tau_1 \ldots \text{cons}_n : \tau_n
\]

\[
e \in \text{Exp} ::= x | n | b | \text{nil} | (x_1 : \tau_1 \ldots x_n : \tau_n) \Rightarrow e | e_1 (e_1 \ldots e_n) \ |
\]

\[
| \text{if } e_1 \ e_2 \ e_3 | \text{let } x = e_1 \text{ in } e_2 | \text{rec } x = e_1 \text{ in } e_2 | [\text{fld}_1 = e_1 \ldots \text{fld}_n = e_n] \\
| e.\text{fld} | \text{cons}(\tau_1 \ldots \tau_k) e | \text{case } e \text{ of } \text{cons}_1 x_1 \Rightarrow e_1 \ldots \text{cons}_n x_n \Rightarrow e_n \\
| [T_1 \ldots T_k] \Rightarrow e | e(\tau_1 \ldots \tau_k)
\]

Compared to the SimpleFUN language in handout 4, we have performed the following changes to get the PolyFUN language above:

- Add type polymorphism to variants, which now act like generics. User-defined variant types now include declarations of type variables which can be used in the constructor types: \text{type name}[T_1 \ldots T_k] = \text{cons}_1 : \tau_1 \ldots \text{cons}_n : \tau_n instead of just \text{type name} = \text{cons}_1 : \tau_1 \ldots \text{cons}_n : \tau_n, where types \tau_1 \ldots \tau_n can now use the type variables \(T_1 \ldots T_k\). Because of this polymorphism, when we construct a variant we need to pass in type arguments to replace the type variables, i.e., \text{cons}(\tau_1 \ldots \tau_k) e instead of just \text{cons} e.

- Add type abstraction and type application to get parametric polymorphism. Type abstraction creates a function whose parameters are type variables (i.e., \([T_1 \ldots T_k] \Rightarrow e\)), and type application calls a type abstraction like a function but passes in types to replace the type variables (i.e., \(e(\tau_1 \ldots \tau_k)\)).

2 PolyFUN Type System

The PolyFUN types are similar to SimpleFUN types with a few changes:

\[
\tau \in \text{Type} = \text{num} | \text{bool} | \text{unit} | (\tau_1 \ldots \tau_n) \rightarrow \tau | [\text{fld}_1 : \tau_1 \ldots \text{fld}_n : \tau_n] | \text{name}[\tau_1 \ldots \tau_k] | T | [T_1 \ldots T_k] \rightarrow \tau
\]

The first five types haven’t changed; the last three are different:

- User-defined variant names are now type constructors rather than types themselves. In other words, \text{name} by itself is not a type—it is a type constructor that takes types as arguments and returns a type as a result: \text{name}(\tau_1 \ldots \tau_k).

- We now have type variables. These variables are introduced by the type abstractions ([\(T_1 \ldots T_k \Rightarrow e\)] and by the variant type declarations (\text{type name}[T_1 \ldots T_k] = \text{cons}_1 : \tau_1 \ldots \text{cons}_n : \tau_n).

- Finally, type abstractions yield a polymorphic type, i.e., a type where the type variables can be replaced with any given type to yield a new type.

The type rules for PolyFUN are exactly like the type rules for SimpleFUN except (1) changes to the \(\text{tdI}\) and \(\text{rmE}\) rules to account for polymorphic variants (recall that the notation \(z[x \mapsto y]\) means to create a copy of \(z\) where every instance of \(x\) has been replaced by \(y\)):

\[
\text{type name}[T_1 \ldots T_k] = \ldots \text{cons} : \tau \ldots \in \text{TypeDef} \quad \Gamma \vdash e : \tau[T_1 \mapsto \tau_1 \ldots T_k \mapsto \tau_k] \\
\Gamma \vdash \text{cons}(\tau_1 \ldots \tau_k) e : \text{name}(\tau_1 \ldots \tau_k)
\]  

(\text{tdI})
\[ \Gamma \vdash e : \text{name}(\tau_1 \ldots \tau_k) \quad \text{type name}[T_1 \ldots T_k] = \text{cons}_1 : \tau_{k+1} \ldots \text{cons}_n : \tau_{k+n} \in \text{TypeDef} \]

\[ \Gamma, x_1 : \tau_{k+1}[T_1 \mapsto \tau_1 \ldots T_k \mapsto \tau_k] \vdash e_1 : \tau \ldots \Gamma, x_n : \tau_{k+n}[T_1 \mapsto \tau_1 \ldots T_k \mapsto \tau_k] \vdash e_n : \tau \quad \text{(TD-E)} \]

And the addition of \text{tabs} and \text{tapp} rules to account for parametric polymorphism:

\[ \Gamma \vdash e : \tau \quad \Gamma \vdash [T_1 \ldots T_k] \Rightarrow e : [T_1 \ldots T_k] \rightarrow \tau \quad \text{\textbf{TABS}} \]

\[ \Gamma \vdash e : [T_1 \ldots T_k] \rightarrow \tau \quad \Gamma \vdash e(\tau_1 \ldots \tau_k) : \tau[T_1 \mapsto \tau_1 \ldots T_k \mapsto \tau_k] \quad \text{\textbf{TAPP}} \]