□ Types

Some views of types:

- A Waste Of Time
  - once popular attitude, but starting to go out of style
  - not very imaginative

- An Abstraction Of The Program
  - like viewing a program from afar
  - different type systems give different views

- A Flow Analysis Of The Program
  - like a map of the algorithm
  - usual notion of types gives crude analysis

□ Benefits of Types

- Catch some dumb mistakes.
- Get certain runtime *guarantees*.
- Help structure your program.
- Make some algorithmic structure explicit.
- Type declarations provide minimal, useful documentation.

□ Simple Type System

$$\tau \;::=\; \mathsf{Bool} \mid \mathsf{Int} \mid \mathsf{List}\,\tau \mid \tau \to \tau \mid (\tau, \tau) \mid ()$$

Bool, Int, () are base types.

List, $\to$, $(,)$ are type constructors.

Example types:

$$(() \to ()) \to () \to ()$$

$$(\mathsf{Bool}, (\mathsf{Int}, \mathsf{Int})) \to \mathsf{Int}$$

□ Typing Judgements

$$\Gamma \vdash e :: \tau$$

usually read as: e has type τ in context Γ.

e is an expression and τ is the type of e.

Γ is a typing context assigning types to variables.

$$\Gamma ::= \cdot \mid x :: \tau, \Gamma$$

We always assume each variable in Γ is unique.
We usually omit $\cdot$

□ Typing Derivations

$$\frac{x :: \tau \in \Gamma}{\Gamma \vdash x :: \tau}\ \mathsf{tp\_var}_x$$

$$\frac{\Gamma, x :: \tau_1 \vdash e :: \tau_2}{\Gamma \vdash \lambda x.e :: \tau_1 \to \tau_2}\ \mathsf{tp\_lam} \qquad \frac{\Gamma \vdash e_1 :: \tau_2 \to \tau_1 \quad \Gamma \vdash e_2 :: \tau_2}{\Gamma \vdash e_1\, e_2 :: \tau_1}\ \mathsf{tp\_app}$$

$$\frac{\Gamma \vdash e_1 :: \tau_1 \quad \Gamma \vdash e_2 :: \tau_2}{\Gamma \vdash (e_1, e_2) :: (\tau_1, \tau_2)}\ \mathsf{tp\_pair}$$

$$\frac{\Gamma \vdash e :: (\tau_1, \tau_2)}{\Gamma \vdash \mathsf{fst}\, e :: \tau_1}\ \mathsf{tp\_fst} \qquad \frac{\Gamma \vdash e :: (\tau_1, \tau_2)}{\Gamma \vdash \mathsf{snd}\, e :: \tau_2}\ \mathsf{tp\_snd}$$

$$\frac{\Gamma, x :: \tau \vdash e :: \tau}{\Gamma \vdash \mathsf{fix}\, x.e :: \tau}\ \mathsf{tp\_fix}$$

□ Typing Derivations contd.

$$\frac{}{\Gamma \vdash \mathsf{tt} :: \mathsf{Bool}}\,\mathsf{tp\_tt} \qquad \frac{}{\Gamma \vdash \mathsf{ff} :: \mathsf{Bool}}\,\mathsf{tp\_ff}$$

$$\frac{\Gamma \vdash e_1 :: \mathsf{Bool} \quad \Gamma \vdash e_2 :: \tau \quad \Gamma \vdash e_3 :: \tau}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 :: \tau}\,\mathsf{tp\_if}$$

$$\frac{}{\Gamma \vdash \mathsf{nil} :: \mathsf{List}\,\tau}\,\mathsf{tp\_nil}_\tau \qquad \frac{\Gamma \vdash e_1 :: \tau \quad \Gamma \vdash e_2 :: \mathsf{List}\,\tau}{\Gamma \vdash e_1 : e_2 :: \mathsf{List}\,\tau}\,\mathsf{tp\_cons}$$

$$\frac{\Gamma \vdash e_1 :: \mathsf{List}\,\tau_1 \quad \Gamma \vdash e_2 :: \tau_2 \quad \Gamma \vdash e_3 :: \tau_1 \to \mathsf{List}\,\tau_1 \to \tau_2}{\Gamma \vdash \mathsf{lcase}\ e_1\ \mathsf{of}\ (e_2, e_3) :: \tau_2}\,\mathsf{tp\_lcase}$$

$$\frac{\Gamma \vdash e_1 :: \mathsf{Int} \quad \Gamma \vdash e_2 :: \mathsf{Int}}{\Gamma \vdash e_1 + e_2 :: \mathsf{Int}}\,\mathsf{tp\_+} \qquad \frac{}{\Gamma \vdash 0 :: \mathsf{Int}}\,\mathsf{tp\_0} \quad \ldots$$

□ Typing Derivation Example

$$
\begin{array}{rcl}
\mathsf{append} & \triangleq & \mathsf{fix}\, r.\lambda x.\lambda y.\mathsf{lcase}\ x\ \mathsf{of}\ (y,\ \lambda h.\lambda t.h : (r\, t\, y)) \\
\tau_a & \triangleq & \mathsf{List\,Int} \to \mathsf{List\,Int} \to \mathsf{List\,Int} \\
\Gamma & \triangleq & r :: \tau_a, x :: \mathsf{List\,Int}, y :: \mathsf{List\,Int} \\
\Gamma' & \triangleq & r :: \tau_a, x :: \mathsf{List\,Int}, y :: \mathsf{List\,Int}, h :: \mathsf{Int}, t :: \mathsf{List\,Int}
\end{array}
$$

$\mathcal{D}_2 =$

$$
\dfrac{\dfrac{\dfrac{}{\Gamma' \vdash r\ ::\ \tau_a}\,\mathsf{tp\_var}_r \quad \dfrac{}{\Gamma' \vdash t\ ::\ \mathsf{List\,Int}}\,\mathsf{tp\_var}_t}{\Gamma' \vdash r\, t\ ::\ \mathsf{List\,Int} \to \mathsf{List\,Int}}\,\mathsf{tp\_app} \quad \dfrac{}{\Gamma' \vdash y\ ::\ \mathsf{List\,Int}}\,\mathsf{tp\_var}_y}{\Gamma' \vdash r\, t\, y\ ::\ \mathsf{List\,Int}}\,\mathsf{tp\_app}
$$

$\mathcal{D}_1 =$

$$
\dfrac{\dfrac{\dfrac{}{\Gamma' \vdash h\ ::\ \mathsf{Int}}\,\mathsf{tp\_var}_h \quad \mathcal{D}_2}{\Gamma, h :: \mathsf{Int}, t :: \mathsf{List\,Int} \vdash h : (r\, t\, y)\ ::\ \mathsf{List\,Int}}\,\mathsf{tp\_cons}}{\Gamma \vdash \lambda h.\lambda t.h : (r\, t\, y)\ ::\ \mathsf{Int} \to \mathsf{List\,Int} \to \mathsf{List\,Int}}\,\mathsf{tp\_lam}^2
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash x\ ::\ \mathsf{List\,Int}}\,\mathsf{tp\_var}_x \quad \dfrac{}{\Gamma \vdash y\ ::\ \mathsf{List\,Int}}\,\mathsf{tp\_var}_y \quad \mathcal{D}_1}{r :: \tau_a, x :: \mathsf{List\,Int}, y :: \mathsf{List\,Int} \vdash \mathsf{lcase}\ x\ \mathsf{of}\ (y,\ \lambda h.\lambda t.h : (r\, t\, y))\ ::\ \mathsf{List\,Int}}\,\mathsf{tp\_lcase}}{r :: \tau_a \vdash \lambda x.\lambda y.\mathsf{lcase}\ x\ \mathsf{of}\ (y,\ \lambda h.\lambda t.h : (r\, t\, y))\ ::\ \tau_a}\,\mathsf{tp\_lam}^2}{\cdot \vdash \mathsf{append}\ ::\ \tau_a}\,\mathsf{tp\_fix}
$$

□ A Limitation

We derived:

$$\cdot \vdash \text{append} :: \text{List}\,\text{Int} \to \text{List}\,\text{Int} \to \text{List}\,\text{Int}$$

But, we could just as easily have derived:

$$\cdot \vdash \text{append} :: \text{List}\,\tau \to \text{List}\,\tau \to \text{List}\,\tau$$

for any choice of τ.

But the expression:

```
(\a a [(1,[True])] [(3,a [True] [True])]) append
```

is not typeable in the previous system.

□ Explicit Types

Remove ambiguity by adding type information to some expressions.

$$e ::= \ldots \mid \lambda x_\tau.e \mid \ldots \mid \mathsf{nil}_\tau \mid \ldots$$

$$\frac{\Gamma, x :: \tau_1 \vdash e :: \tau_2}{\Gamma \vdash \lambda x_{\tau_1}.e :: \tau_1 \to \tau_2}\ \mathsf{tp\_lam} \qquad \frac{}{\Gamma \vdash \mathsf{nil}_\tau :: \mathsf{List}\,\tau}\ \mathsf{tp\_nil}$$

Examples:

$$\mathsf{append} \triangleq \mathsf{fix}\, r.\lambda x_{\mathsf{List\,Int}}.\lambda y_{\mathsf{List\,Int}}.\ \mathsf{lcase}\ x\ \mathsf{of}\ (y,\ \lambda h_{\mathsf{Int}}.\lambda t_{\mathsf{List\,Int}}.h : (r\, t\, y))$$

$$\mathsf{map} \triangleq \mathsf{fix}\, r.\lambda f_{\mathsf{Int}\to\mathsf{Bool}}.\lambda x_{\mathsf{List\,Int}}.\ \mathsf{lcase}\ x\ \mathsf{of}\ (\mathsf{nil}_{\mathsf{Bool}},\ \lambda h_{\mathsf{Int}}.\lambda t_{\mathsf{List\,Int}}.(f\, h) : (r\, f\, t))$$

□ Explicit Parametric Polymorphism

Add polymorphic expressions and types:

$$\begin{array}{rcl} e & ::= & \ldots \mid \Lambda\alpha.e \mid e\,[\tau] \\ \tau & ::= & \ldots \mid \forall\alpha.\tau \end{array}$$

where α is a type variable.

$\forall$ binds type variable α so α is not free in $\forall\alpha.\tau$.

Polymorphic typing rules:

$$\frac{\Gamma \vdash e :: \tau}{\Gamma \vdash \Lambda\alpha.e :: \forall\alpha.\tau}\ \text{tp\_tlam} \qquad \frac{\Gamma \vdash e :: \forall\alpha.\tau}{\Gamma \vdash e[\tau'] :: \tau[\alpha := \tau']}\ \text{tp\_tapp}$$

where α is not free in Γ in tp_tlam.

□ Explicit Parametric Polymorphism

Examples

Evaluation for polymorphic expressions:

$$\frac{e_1 \hookrightarrow \Lambda\alpha.e_2 \quad e_2[\alpha := \tau] \hookrightarrow v}{e_1[\tau] \hookrightarrow v}$$

Examples:

$$\text{append} \triangleq \text{fix}\, r.\Lambda\alpha.\lambda x_{\mathsf{List}\,\alpha}.\lambda y_{\mathsf{List}\,\alpha}.\ \text{lcase}\ x\ \text{of}\ (y,\ \lambda h_\alpha.\lambda t_{\mathsf{List}\,\alpha}.h : (r\,[\alpha]\,t\,y))$$

$$\text{map} \triangleq \text{fix}\, r.\Lambda\alpha.\Lambda\beta.\lambda f_{\alpha\to\beta}.\lambda x_{\mathsf{List}\,\alpha}.\ \text{lcase}\ x\ \text{of}\ (\text{nil}_\beta,\ \lambda h_\alpha.\lambda t_{\mathsf{List}\,\alpha}.(f\,h) : (r\,[\alpha]\,[\beta]\,f\,t))$$