iSAQB Advanced DSL - Types

Michael Sperber

Created: 2025-07-21 Mon 10:41

(LG 4-2) Types

- **types**: *syntactic* mechanism for classifying values
- (static) **type system**: assign types to program locations
- **dynamic types** = "**untyped**" - classification happens at runtime
- **type soundness**: each expression evaluates to member of its type
- **gradual typing**: only some expressions have a type

Simple Types

$$\begin{array}{rcll} \langle T \rangle & \rightarrow & \langle B_1 \rangle \mid \ldots \mid \langle B_n \rangle & \text{base types} \\ & \mid & \langle T \rangle \rightarrow \langle T \rangle & \end{array}$$

$$\tau \in T$$

$\Gamma : X \rightarrow V$ typing environment

$\Gamma[x \mapsto \tau]$ environment extension

The Simply Typed Lambda Calculus

$$\frac{\tau = \Gamma(x)}{\Gamma \vdash x : \tau}$$

$$\frac{F : B_{i_1} \times \ldots \times B_{i_n} \rightarrow B_{i_F} \qquad \Gamma \vdash e_j : B_{i_j}}{\Gamma \vdash F\, e_1 \; \ldots \; e_n : B_{i_F}}$$

$$\frac{\Gamma[x \mapsto \tau_1] \vdash e : \tau_2}{\Gamma \vdash \lambda x.\, e : \tau_1 \rightarrow \tau_2}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\; e_2 : \tau_2}$$

(LG 4-2) Hindley-Milner Type System: Language

$$
\begin{array}{rcl}
\langle \mathcal{L}_{\mathrm{HM}} \rangle \rightarrow & & \langle X \rangle \\
& \mid & \langle \mathcal{L}_{\mathrm{HM}} \rangle\, \langle \mathcal{L}_{\mathrm{HM}} \rangle \\
& \mid & \lambda\, \langle V \rangle .\, \langle \mathcal{L}_{\mathrm{HM}} \rangle \\
& \mid & \texttt{let}\ \langle V \rangle = \langle \mathcal{L}_{\mathrm{HM}} \rangle\ \mathrm{in}\ \langle \mathcal{L}_{\mathrm{HM}} \rangle
\end{array}
$$

Hindley-Milner Type System: Type Language

Monotypes:

$$
\begin{array}{rcll}
\langle T \rangle \rightarrow & & \langle A \rangle & \text{type variables} \\
& | & \langle C \rangle \ \langle T \rangle \ \ldots \ \langle T \rangle & \text{type-constructor applications}
\end{array}
$$

Polytypes:

$$
\begin{array}{rcl}
\langle P \rangle \rightarrow & & \langle T \rangle \\
& | & \forall \langle A \rangle . \langle P \rangle
\end{array}
$$

Hindley-Milner Type System: Auxiliary Definitions

Typing context: $\langle G\rangle \to \epsilon \mid \langle G\rangle, \langle V\rangle : \langle P\rangle$

Free type variables:

$$
\begin{array}{rcl}
\mathrm{free}(\alpha) & = & \{\alpha\} \\
\mathrm{free}(C\ \tau_1 \ldots \tau_n) & = & \bigcup_{i=1}^{n} \mathrm{free}(\tau_i) \\
\mathrm{free}(\Gamma) & = & \bigcup_{x:\sigma\in\Gamma} \mathrm{free}(\sigma) \\
\mathrm{free}(\forall\alpha.\,\sigma) & = & \mathrm{free}(\sigma) \setminus \{\alpha\} \\
\mathrm{free}(\gamma \vdash e : \sigma) & = & \mathrm{free}(\sigma) \setminus \mathrm{free}(\Gamma)
\end{array}
$$

Generalization:

$$\overline{\Gamma}(\tau) = \forall\hat{\alpha}.\,\tau, \hat{\alpha} = \mathrm{free}(\tau) \setminus \mathrm{free}(\Gamma)$$

Hindley-Milner Type System: Subtyping

$$\frac{\tau' = \tau[\alpha_i \mapsto \tau_i] \qquad \beta_i \notin \mathrm{free}(\forall\alpha_1 \ldots \forall\alpha_n.\,\tau)}{\forall\alpha_1 \ldots \forall\alpha_n.\,\tau \sqsubseteq \forall\beta_1 \ldots \forall\beta_m.\,\tau'}$$

HM Type System: Syntax-Directed

$$\frac{x : \sigma \in \Gamma \qquad \sigma \sqsubseteq \tau}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau \to \tau' \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ e_2 : \tau'}$$

$$\frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda x.\, e : \tau \to \tau'}$$

$$\frac{e_1 : \tau \qquad \Gamma, x : \overline{\Gamma}(\tau) \vdash e_2 : \tau'}{\Gamma \vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 : \tau'}$$

Hindley-Milner Type System: Properties

- algorithm for *type inference*
- uses *unification*
- finds *most general type*
- basis for Standard ML, OCaml, F#, Haskell, ...

Exercise: Type System for Tim

- In what ways can a Tim program go wrong?
- Design a type language for Tim.
- What changes to the language are necessary to support types?
- Design a type system for Tim.
- Implement a type checker for Tim.

Exercise sketch: Type System for Tim

- differentiate base types (coordinates, direction, format)
- for formats, determine their *extent* i.e. how tall/wide a format is, and it what direction it stretches indefinitely
- use extend to determine whether extents in a record or list overlap

(LG 4-2) Monads for Overloading in Haskell

Check out ../haskell-code/Specs.hs.

(LG 4-2) Overloading of Numerical Constants and Operations

```
class Num a where
  (+) :: a -> a -> a
  (-) :: a -> a -> a
  (*) :: a -> a -> a
  negate :: a -> a
  abs :: a -> a
  signum :: a -> a
  fromInteger :: Integer -> a
```

Exercise: Effects in "Expressions"

How could effectful computations ("read sensor") be integrated as part of expressions?