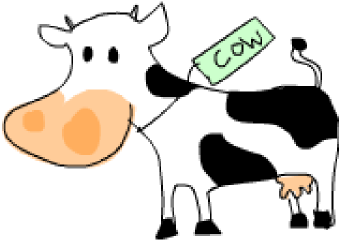
Last time on **Types**...



picture from http://learnyouahaskell.com

Last time on **Types**...

- Simply-typed λ-calculus (recap)

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

- Parametric polymorphism

$$\mathtt{let}\, f = \lambda x(x)\, \mathtt{in}\, (f\, \mathtt{true}) :: (f\, \mathtt{nil})$$

- The beginnings of the Mini-ML type system...

  let-*polymorphism*

Mini-ML types and type schemes

Types

$$
\begin{array}{rcll}
\tau & ::= & \alpha & \text{type variable} \\
& | & \mathit{bool} & \text{type of booleans} \\
& | & \tau \to \tau & \text{function type} \\
& | & \tau\ \mathit{list} & \text{list type}
\end{array}
$$

where α ranges over a fixed, countably infinite set TyVar.

Type Schemes

$$\sigma \quad ::= \quad \forall\, A\,(\tau)$$

where A ranges over finite subsets of the set TyVar.

When $A = \{\alpha_1, \ldots, \alpha_n\}$, we write $\forall\, A\,(\tau)$ as

$$\forall\, \alpha_1, \ldots, \alpha_n\,(\tau).$$

The 'generalises' relation between type schemes and types

We say a type scheme $\sigma = \forall\, \alpha_1, \ldots, \alpha_n\, (\tau')$ generalises a type τ, and write $\boxed{\sigma \succ \tau}$ if τ can be obtained from the type τ' by simultaneously substituting some types τ_i for the type variables α_i $(i = 1, \ldots, n)$:

$$\tau = \tau'[\tau_1/\alpha_1, \ldots, \tau_n/\alpha_n].$$

(N.B. The relation is unaffected by the particular choice of names of bound type variables in σ.)

The converse relation is called specialisation: a type τ is a specialisation of a type scheme σ if $\sigma \succ \tau$.

Generalisations: some examples and non-examples

- $\forall\alpha.(\alpha \to \alpha) \succ bool \to bool$ with $[bool/\alpha]$
- $\forall\alpha.(\alpha \to \alpha) \not\succ (int \to bool)$
- $\forall\alpha.(\alpha \to \alpha) \succ [\beta] \to [\beta]$ with $[[\beta]/\alpha]$
- $\forall\alpha,\beta.(\alpha \to \beta) \succ (int \to bool)$ with $[int/\alpha, bool/\beta]$
- $\forall\alpha.(\alpha \to \beta) \not\succ (int \to bool)$
- $\forall\alpha.(\alpha \to \beta) \succ (int \to \beta)$ with $[int/\alpha]$

Mini-ML typing judgement

takes the form $\boxed{\Gamma \vdash M : \tau}$ where

- the typing environment Γ is a finite function from variables to *type schemes*.
  (We write $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ to indicate that Γ has domain of definition $dom(\Gamma) = \{x_1, \ldots, x_n\}$ and maps each x_i to the type scheme σ_i for $i = 1..n$.)

- M is a Mini-ML expression

- τ is a Mini-ML type.

Mini-ML expressions, M

$::=$	x	variable
$\mid$	`true`	boolean values
$\mid$	`false`	
$\mid$	`if` M `then` M `else` M	conditional
$\mid$	$\lambda x(M)$	function abstraction
$\mid$	$M\ M$	function application
$\mid$	`let` $x = M$ `in` M	local declaration
$\mid$	`nil`	nil list
$\mid$	$M :: M$	list cons
$\mid$	`case` M `of` `nil` $\Rightarrow M \mid x :: x \Rightarrow M$	case expression

Mini-ML type system, I

$$\Gamma \vdash x : \tau \quad \text{if } (x : \sigma) \in \Gamma \quad \text{and } \sigma \succ \tau \qquad (\mathrm{var} \succ)$$

$$\Gamma \vdash B : bool \quad \text{if } B \in \{\mathtt{true}, \mathtt{false}\} \qquad (\mathrm{bool})$$

$$\frac{\Gamma \vdash M_1 : bool \qquad \Gamma \vdash M_2 : \tau \qquad \Gamma \vdash M_3 : \tau}{\Gamma \vdash \mathtt{if}\ M_1\ \mathtt{then}\ M_2\ \mathtt{else}\ M_3 : \tau} \quad (\mathrm{if})$$

Mini-ML type system, II

$$\Gamma \vdash \mathtt{nil} : \tau\ \mathit{list} \tag{nil}$$

$$\frac{\Gamma \vdash M_1 : \tau \qquad \Gamma \vdash M_2 : \tau\ \mathit{list}}{\Gamma \vdash M_1 :: M_2 : \tau\ \mathit{list}} \tag{cons}$$

$$\frac{\Gamma \vdash M_1 : \tau_1\ \mathit{list} \qquad \Gamma \vdash M_2 : \tau_2 \qquad \Gamma, x_1 : \tau_1, x_2 : \tau_1\ \mathit{list} \vdash M_3 : \tau_2}{\Gamma \vdash \mathtt{case}\ M_1\ \mathtt{of}\ \mathtt{nil} \Rightarrow M_2 \mid x_1 :: x_2 \Rightarrow M_3 : \tau_2} \quad \text{if } x_1, x_2 \notin \mathit{dom}(\Gamma) \wedge x_1 \neq x_2 \tag{case}$$

Mini-ML type system, III

$$\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x(M) : \tau_1 \to \tau_2} \quad \text{if } x \notin dom(\Gamma) \qquad \text{(fn)}$$

$$\frac{\Gamma \vdash M_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash M_2 : \tau_1}{\Gamma \vdash M_1\, M_2 : \tau_2} \qquad \text{(app)}$$

$$\frac{\Gamma \vdash M_1 : \tau \qquad \Gamma, x : \forall\, A\,(\tau) \vdash M_2 : \tau'}{\Gamma \vdash \mathtt{let}\, x = M_1 \,\mathtt{in}\, M_2 : \tau'} \quad \begin{array}{l} \text{if } x \notin dom(\Gamma) \\ \wedge\ A = ftv(\tau) - ftv(\Gamma) \end{array} \qquad \text{(let)}$$

Assigning type schemes to Mini-ML expressions

Given a type scheme $\sigma = \forall\, A\,(\tau)$, write

$$\boxed{\Gamma \vdash M : \sigma}$$

if $A = \mathit{ftv}(\tau) - \mathit{ftv}(\Gamma)$ and $\Gamma \vdash M : \tau$ is derivable from the axiom and rules on Slides 65–67.

When $\Gamma = \{\,\}$ we just write $\boxed{\vdash M : \sigma}$ for $\{\,\} \vdash M : \sigma$ and say that the (necessarily closed—see Exercise 2) expression M is *typeable* in Mini-ML with type scheme σ.

Mini-ML - Type checking, typeability, and type inference

- Type-checking problem: given **closed** M, and σ, is $\{\} \vdash M : \sigma$ derivable in the type system?
- Typeability problem: given **closed** M, is there any σ for which $\{\} \vdash M : \sigma$ is derivable in the type system?

Two examples involving self-application

$$M \stackrel{\text{def}}{=} \texttt{let}\, f = \lambda x_1(\lambda x_2(x_1))\,\texttt{in}\, f\, f$$

$$M' \stackrel{\text{def}}{=} (\lambda f(f\, f))\, \lambda x_1(\lambda x_2(x_1))$$

Are M and M' typeable in the Mini-ML type system?

Example using let polymorphism

(z is used polymorphically)

$$\Gamma = y:\beta,\ z:\forall\tau.(\tau\to\tau\to bool)$$

$\forall A. \ldots$

$$A = ftv(\tau) - ftv(\Gamma) = \{\alpha,\beta\} - \{\beta\} = \{\alpha\}$$

$$\frac{\overbrace{x:\forall\alpha.\alpha\to\beta,\Gamma}^{\Gamma'} \vdash x:\beta\to\beta \quad (\text{var}) \qquad \Gamma'\vdash y:\beta}{\Gamma'\vdash (x\ y):\beta}$$

$$\frac{\vdash z:\beta\to\beta\to bool \qquad \Gamma'\vdash (x\ y):\beta}{\Gamma'\vdash z\,(x\ y)}$$

$$\frac{\Gamma'\vdash nil:[\gamma] \qquad \Gamma'\vdash x:[\gamma]\to\beta}{\Gamma'\vdash x\ nil:\beta}$$

$$\frac{u:\alpha_1,\ z,\ldots,\ y:\beta\vdash y:\beta}{z\ldots,y:\ldots\vdash \lambda u.y:\alpha\to\beta}$$

$$\frac{\Gamma'\vdash z\,(x\ y) \qquad \Gamma'\vdash x\ nil:\beta}{x:\forall\alpha.\alpha\to\beta,\Gamma\vdash (z\,(x\ y))\,(x\ nil):bool}$$

$$\frac{z\ldots,y:\ldots\vdash \lambda u.y:\alpha\to\beta \qquad x:\forall\alpha.\alpha\to\beta,\Gamma\vdash (z\,(x\ y))\,(x\ nil):bool}{\Gamma\vdash \texttt{let}\ x=\lambda u.y\ \texttt{in}\ (z\,(x\ y))\,(x\ nil):bool}$$