# Mini-ML expressions

$$
\begin{array}{rcll}
M & ::= & x & \text{variable} \\
& | & \texttt{true} & \text{boolean values} \\
& | & \texttt{false} & \\
& | & \texttt{if}\ M\ \texttt{then}\ M\ \texttt{else}\ M & \text{conditional} \\
& | & \lambda x\,(M) & \text{function abstraction} \\
& | & M\,M & \text{function application} \\
& | & \texttt{let}\ x = M\ \texttt{in}\ M & \text{local declaration} \\
& | & \texttt{nil} & \text{nil list} \\
& | & M :: M & \text{list cons} \\
& | & \texttt{case}\ M\ \texttt{of}\ \texttt{nil} \Rightarrow M \mid x :: x \Rightarrow M & \text{case expression}
\end{array}
$$

(abstract syntax <u>trees</u>)

# Mini-ML types and type schemes

*Types*

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

# Mini-ML types and type schemes

*Types*

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

where $\alpha$ ranges over a fixed, countably infinite set $\mathbf{TyVar}$.

# Mini-ML types and type schemes

*Types*

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

where $\alpha$ 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**.

# Mini-ML types and type schemes

*Types*

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

where $\alpha$ 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 $A = \{\alpha_1, \ldots, \alpha_n\}$ (mutually distinct type variables) we write $\forall A\,(\tau)$ as

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

E.g.s of type schemes:

$\forall \alpha, \beta\,(\alpha \to \beta)$ $\quad \forall \alpha\,(\alpha\, list \to \beta)$

# Mini-ML types and type schemes

*Types* $\quad \tau ::= \alpha$ type variable
$\quad\quad | \ bool$ type of booleans
$\quad\quad | \ \tau \to \tau$ function type
$\quad\quad | \ \tau \ list$ list type

where $\alpha$ ranges over a fixed, countably infinite set $\mathbf{TyVar}$.

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

where $A$ ranges over finite subsets of the set $\mathbf{TyVar}$.

possibly empty, e.g.

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

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

E.g.s of type schemes:

$\forall \alpha, \beta \, (\alpha \to \beta) \quad \forall \alpha \, (\alpha \ list \to \beta) \quad \forall \{\} (\alpha \to bool)$

# Mini-ML types and type schemes

*Types*

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

where $\alpha$ ranges over a fixed, countably infinite set $\mathbf{TyVar}$.

*Type Schemes*

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

where $A$ ranges over finite subsets of the set $\mathbf{TyVar}$.

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

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

When $A = \{\}$ is empty, we write $\forall A\,(\tau)$ just as $\tau$. In other words, **we regard the set of types as a subset of the set of type schemes by identifying the type $\tau$ with the type scheme $\forall\{\,\}\,(\tau)$.**

# Mini-ML typing judgement

takes the form

$$\Gamma \vdash M : \tau$$

where

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

# Mini-ML type system, I

$$(\mathbf{var}\succ)\ \frac{}{\Gamma \vdash x : \tau}\ \text{if } (x : \sigma) \in \Gamma \text{ and } \sigma \succ \tau$$

$$(\mathbf{bool})\ \frac{}{\Gamma \vdash B : bool}\ \text{if } B \in \{\mathtt{true}, \mathtt{false}\}$$

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

# Specialising type schemes to types

A type $\tau$ is a *specialisation* of a type scheme $\sigma = \forall \alpha_1, \ldots, \alpha_n\,(\tau')$ if $\tau$ can be obtained from the type $\tau'$ by simultaneously substituting some types $\tau_i$ for the type variables $\alpha_i$ ($i = 1, \ldots, n$):

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

In this case we write $\boxed{\sigma \succ \tau}$

E.g. $\forall \alpha, \beta\,(\alpha \to \beta) \succ \beta \to bool$

& $\forall \alpha\,(\alpha \to \beta) \succ bool \to \beta$

but $\forall \alpha\,(\alpha \to \beta) \not\succ bool \to bool$

# Specialising type schemes to types

A type $\tau$ is a *specialisation* of a type scheme $\sigma = \forall \alpha_1, \ldots, \alpha_n\,(\tau')$ if $\tau$ can be obtained from the type $\tau'$ by simultaneously substituting some types $\tau_i$ for the type variables $\alpha_i$ $(i = 1, \ldots, n)$:

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

In this case we write $\boxed{\sigma \succ \tau}$

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

Identify type schemes up to renaming $\forall$-bound type variables, e.g.

$$\forall \alpha\,(\alpha \to \beta) = \forall \gamma\,(\gamma \to \beta) \neq \forall \gamma\,(\gamma \to \gamma)$$

# Specialising type schemes to types

A type $\tau$ is a *specialisation* of a type scheme $\sigma = \forall \alpha_1, \ldots, \alpha_n\,(\tau')$ if $\tau$ can be obtained from the type $\tau'$ by simultaneously substituting some types $\tau_i$ for the type variables $\alpha_i$ $(i = 1, \ldots, n)$:

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

In this case we write $\boxed{\sigma \succ \tau}$

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

The converse relation is called *generalisation*: a type scheme $\sigma$ generalises a type $\tau$ if $\sigma \succ \tau$.

# Mini-ML type system, III

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

$$(\mathbf{app})\ \frac{\Gamma \vdash M : \tau_1 \to \tau_2 \qquad \Gamma \vdash N : \tau_1}{\Gamma \vdash M\,N : \tau_2}$$

# Mini-ML type system, III

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

$$(\mathbf{app})\ \frac{\Gamma \vdash M : \tau_1 \to \tau_2 \qquad \Gamma \vdash N : \tau_1}{\Gamma \vdash M\,N : \tau_2}$$

abbreviation for $x : \forall \{\}\, \tau_1$

# Mini-ML type system, II

$$(\mathbf{nil})\ \frac{}{\Gamma \vdash \mathtt{nil} : \tau\ \mathit{list}}$$

$$(\mathbf{cons})\ \frac{\Gamma \vdash M : \tau \qquad \Gamma \vdash L : \tau\ \mathit{list}}{\Gamma \vdash M :: L : \tau\ \mathit{list}}$$

$$(\mathbf{case})\ \frac{\begin{array}{c}\Gamma \vdash L : \tau\ \mathit{list} \qquad \Gamma \vdash N : \tau' \\ \Gamma, x : \tau, \ell : \tau\ \mathit{list} \vdash C : \tau'\end{array}}{\Gamma \vdash (\mathtt{case}\ L\ \mathtt{of}\ \mathtt{nil} \Rightarrow N \mid x :: \ell \Rightarrow C) : \tau'}\ \text{if } x \neq \ell \text{ and } x, \ell \notin \mathit{dom}(\Gamma)$$

# Mini-ML type system, III

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

# Mini-ML type system, III

$$(\mathbf{let})\ \frac{\Gamma \vdash M_1 : \tau \qquad \Gamma, x : \forall A\,(\tau) \vdash M_2 : \tau'}{\Gamma \vdash (\mathtt{let}\ x = M_1\ \mathtt{in}\ M_2) : \tau'}$$ if $x \notin dom(\Gamma)$ and $A = ftv(\tau) - ftv(\Gamma)$

$ftv(\tau)$ = all type variables occuring in $\tau$

$ftv\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\} = ftv(\sigma_1) \cup \cdots \cup ftv(\sigma_n)$

where if $\sigma = \forall A(\tau)$, then $ftv(\sigma) = ftv(\tau) - A$

# Example of using the (**let**) rule

$$(\textbf{let})\ \frac{\Gamma \vdash M_1 : \tau \qquad \Gamma, x : \forall A\,(\tau) \vdash M_2 : \tau'}{\Gamma \vdash (\texttt{let}\,x = M_1\,\texttt{in}\,M_2) : \tau'}\ \text{if } x \notin dom(\Gamma) \text{ and } A = ftv(\tau) - ftv(\Gamma)$$

If $\Gamma \vdash M_1 : \tau$ is $\boxed{y : \beta, z : \forall \gamma\,(\gamma \to \gamma \to bool) \vdash \lambda u\,(y) : \alpha \to \beta}$

# Example of using the (let) rule

$$(\mathbf{let})\ \frac{\Gamma \vdash M_1 : \tau \qquad \Gamma, x : \forall A\,(\tau) \vdash M_2 : \tau'}{\Gamma \vdash (\mathtt{let}\, x = M_1\, \mathtt{in}\, M_2) : \tau'} \ \text{if } x \notin dom(\Gamma) \text{ and } A = ftv(\tau) - ftv(\Gamma)$$

If $\Gamma \vdash M_1 : \tau$ is $\boxed{y : \beta, z : \forall\gamma\,(\gamma \to \gamma \to bool) \vdash \lambda u\,(y) : \alpha \to \beta}$

then $A = \{\alpha, \beta\} - \{\beta\} = \{\alpha\}$ and $\forall A\,(\tau) = \forall\alpha\,(\alpha \to \beta)$.

# Example of using the (let) rule

$$\textbf{(let)}\ \frac{\Gamma \vdash M_1 : \tau \qquad \Gamma, x : \forall A\,(\tau) \vdash M_2 : \tau'}{\Gamma \vdash (\texttt{let}\, x = M_1 \,\texttt{in}\, M_2) : \tau'}\ \text{if } x \notin dom(\Gamma) \text{ and } A = ftv(\tau) - ftv(\Gamma)$$

If $\Gamma \vdash M_1 : \tau$ is $\boxed{y : \beta, z : \forall \gamma\,(\gamma \to \gamma \to bool) \vdash \lambda u\,(y) : \alpha \to \beta}$

then $A = \{\alpha, \beta\} - \{\beta\} = \{\alpha\}$ and $\forall A\,(\tau) = \forall \alpha\,(\alpha \to \beta)$.

So if $\Gamma, x : \forall A\,(\tau) \vdash M_2 : \tau'$ is

$$\boxed{y : \beta, z : \forall \gamma\,(\gamma \to \gamma \to bool), x : \forall \alpha\,(\alpha \to \beta) \vdash z\,(x\,y)\,(x\,\texttt{nil}) : bool}$$

# Example of using the (let) rule

$$\textbf{(let)}\ \frac{\Gamma \vdash M_1 : \tau \qquad \Gamma, x : \forall A\,(\tau) \vdash M_2 : \tau'}{\Gamma \vdash (\mathtt{let}\, x = M_1\, \mathtt{in}\, M_2) : \tau'}\ \text{if } x \notin dom(\Gamma) \text{ and } A = ftv(\tau) - ftv(\Gamma)$$

If $\Gamma \vdash M_1 : \tau$ is $\boxed{y : \beta, z : \forall \gamma\,(\gamma \to \gamma \to bool) \vdash \lambda u\,(y) : \alpha \to \beta}$

then $A = \{\alpha, \beta\} - \{\beta\} = \{\alpha\}$ and $\forall A\,(\tau) = \forall \alpha\,(\alpha \to \beta)$.

So if $\Gamma, x : \forall A\,(\tau) \vdash M_2 : \tau'$ is

$$\boxed{y : \beta, z : \forall \gamma\,(\gamma \to \gamma \to bool), x : \forall \alpha\,(\alpha \to \beta) \vdash z\,(x\,y)\,(x\,\mathtt{nil}) : bool}$$

then applying **(let)** yields

$$\boxed{y : \beta, z : \forall \gamma\,(\gamma \to \gamma \to bool) \vdash \mathtt{let}\, x = \lambda u\,(y)\, \mathtt{in}\, z\,(x\,y)\,(x\,\mathtt{nil}) : bool}$$

# Mini-ML type system, III

**Definition.** We write $\boxed{\Gamma \vdash M : \forall A\,(\tau)}$ to mean $\Gamma \vdash M : \tau$ is derivable from the Mini-ML typing rules and that $A = ftv(\tau) - ftv(\Gamma)$.

# Mini-ML type system, III

$$(\mathbf{let})\ \frac{\Gamma \vdash M_1 : \tau \qquad \Gamma, x : \forall A\,(\tau) \vdash M_2 : \tau'}{\Gamma \vdash (\mathtt{let}\, x = M_1 \,\mathtt{in}\, M_2) : \tau'} \text{ if } x \notin dom(\Gamma) \text{ and } A = ftv(\tau) - ftv(\Gamma)$$

**Definition.** We write $\boxed{\Gamma \vdash M : \forall A\,(\tau)}$ to mean $\Gamma \vdash M : \tau$ is derivable from the Mini-ML typing rules and that $A = ftv(\tau) - ftv(\Gamma)$.

(So $(\mathbf{let})$ is equivalent to $\dfrac{\Gamma \vdash M_1 : \sigma \qquad \Gamma, x : \sigma \vdash M_2 : \tau'}{\Gamma \vdash (\mathtt{let}\, x = M_1 \,\mathtt{in}\, M_2) : \tau'}$ if $x \notin dom(\Gamma)$.)

(Cf. Slide 6)

Mini-ML type-checking problem:

a given $\Gamma$, $M$ & $\sigma$, does $\Gamma \vdash M : \sigma$ hold?

Mini-ML type-inference problem:

b given $\Gamma$ & $M$, does there exist $\sigma$ such ~~that~~ $\Gamma \vdash M : \sigma$ holds?

Solving a entails solving b, because of the form of the (let) typing rule.