Polymorphism of let-bound variables in ML

For example in

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

$\lambda x(x)$ has type $\tau \to \tau$ for any type τ, and the variable f to which it is bound is used polymorphically:

- in $(f\, \texttt{true})$, f has type $bool \to bool$
- in $(f\, \texttt{nil})$, f has type $bool\ list \to bool\ list$

Overall, the expression has type $bool\ list$.

Mini-ML expressions, M

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

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

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

E.g.s of type schemes: $\boxed{\forall \alpha, \beta\,(\alpha \to \beta)}$ $\boxed{\forall \alpha\,(\alpha\,list \to \beta)}$

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 α 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 $\mathbf{TyVar}$. possibly empty

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

eg

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

E.g.s of type schemes: $\boxed{\forall \alpha, \beta\,(\alpha \to \beta)}$ $\boxed{\forall \alpha\,(\alpha\, list \to \beta)}$ $\boxed{\forall \{\}\,(\alpha \to bool)}$

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 an Mini-ML expression
- τ is an Mini-ML type.

Mini-ML type system, I

(var $\succ$) $\Gamma \vdash x : \tau$ if $(x : \sigma) \in \Gamma$ and $\sigma \succ \tau$

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

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

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 $\sigma \succ \tau$.

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

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

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 $\sigma \succ \tau$.

So we identify type schemes up to renaming bound type vars

e.g. $\forall\alpha(\alpha \to \alpha') = \forall\alpha''(\alpha'' \to \alpha') \neq \forall\alpha'(\alpha' \to \alpha')$

Mini-ML type system, II

(nil) $\Gamma \vdash \texttt{nil} : \tau\ list$

(cons) $$\frac{\Gamma \vdash M_1 : \tau \quad \Gamma \vdash M_2 : \tau\ list}{\Gamma \vdash M_1 :: M_2 : \tau\ list}$$

(case) $$\frac{\begin{array}{c}\Gamma \vdash M_1 : \tau_1\ list \qquad \Gamma \vdash M_2 : \tau_2 \\ \Gamma, x_1 : \tau_1, x_2 : \tau_1\ list \vdash M_3 : \tau_2\end{array}}{\begin{array}{c}\Gamma \vdash \mathtt{case}\ M_1\ \mathtt{of}\ \mathtt{nil} \Rightarrow M_2 \\ \mid x_1 :: x_2 \Rightarrow M_3 : \tau_2\end{array}}$$ if $x_1, x_2 \notin dom(\Gamma)$ and $x_1 \neq x_2$

Mini-ML type system, II

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

(cons) $$\frac{\Gamma \vdash M_1 : \tau \quad \Gamma \vdash M_2 : \tau\ list}{\Gamma \vdash M_1 :: M_2 : \tau\ list}$$

(case) $$\frac{\begin{array}{c}\Gamma \vdash M_1 : \tau_1\ list \qquad \Gamma \vdash M_2 : \tau_2 \\ \Gamma, x_1 : \tau_1, x_2 : \tau_1\ list \vdash M_3 : \tau_2\end{array}}{\begin{array}{c}\Gamma \vdash \mathtt{case}\ M_1\ \mathtt{of}\ \mathtt{nil} \Rightarrow M_2 \\ \mid x_1 :: x_2 \Rightarrow M_3 : \tau_2\end{array}}$$ if $x_1, x_2 \notin dom(\Gamma)$ and $x_1 \neq x_2$

abbreviation for $x_1 : \forall \{\} (\tau_1)$

Mini-ML type system, III

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

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

Mini-ML type system, IV

(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 $A = ftv(\tau) - ftv(\Gamma)$

$\boxed{ftv(\tau) = \text{all type vars occurring in type } \tau}$

$\boxed{ftv\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\} = ftv(\sigma_1) \cup \cdots \cup ftv(\sigma_2)}$

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

Example of the (let) rule

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

so A is $\{\alpha, \beta\} - \{\beta\} = \{\alpha\}$

Example of the (let) rule

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

so A is $\{\alpha, \beta\} - \{\beta\} = \{\alpha\}$

$\Gamma, x : \forall A(\tau) \vdash M_2 : \tau'$ is $\boxed{\{y : \beta, z : \forall \gamma(\gamma \to \gamma \to bool), x : \forall \alpha(\alpha \to \beta)\} \vdash z(x\,y)(x\,nil) : bool}$

Example of the (let) rule

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

so A is $\{\alpha,\beta\} - \{\beta\} = \{\alpha\}$

$\Gamma, x:\forall A(\tau) \vdash M_2 : \tau'$ is $\{y:\beta, z:\forall\gamma(\gamma\to\gamma\to bool), x:\forall\alpha(\alpha\to\beta)\} \vdash$
$z(xy)(x\,nil) : bool$

Applying (let) we get

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

Assigning type schemes to Mini-ML expressions

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

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

if $A = ftv(\tau) - ftv(\Gamma)$ and $\Gamma \vdash M : \tau$ is derivable from the axiom and rules on Slides 16–19.

When $\Gamma = \{\,\}$ we just write $\boxed{\vdash M : \sigma}$ for $\{\,\} \vdash M : \sigma$ and say that the (necessarily closed—see Exercise 2.5.2) expression M is *typeable* in Mini-ML with type scheme σ.

["closed" = "has no free variables"]

[Cf. Slide 7]

(a) A Mini-ML type checking problem:

$\boxed{\text{given closed } M \text{ and } \sigma, \text{ does } \vdash M : \sigma \text{ hold?}}$

(b) A Mini-ML typeability problem

$\boxed{\text{given closed } M, \text{ does there exist a closed } \sigma \text{ such that } \vdash M : \sigma \text{ holds?}}$

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

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?