This time on **Types**...

Polymorphic λ-calculus

(polymorphic λ-binding). Let's us type:

$$\lambda f((f\,\texttt{true}) :: (f\,\texttt{nil}))$$

λ-bound variables in ML cannot be used polymorphically within a function abstraction

E.g. $\lambda f((f\,\texttt{true}) :: (f\,\texttt{nil}))$ and $\lambda f(f\,f)$ are not typeable in the ML type system.

Syntactically, because in rule

$$(\mathrm{fn})\ \frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x(M) : \tau_1 \to \tau_2}$$

the abstracted variable has to be assigned a *trivial* type scheme (recall $x : \tau_1$ stands for $x : \forall\,\{\,\}\,(\tau_1)$).

Semantically, because $\forall\,A\,(\tau_1) \to \tau_2$ is not semantically equivalent to an ML type when $A \neq \{\,\}$.

$$\dfrac{\dfrac{\dfrac{}{f:\forall\emptyset.\tau_2 \vdash f:\tau_4}\,(\text{vars}) \qquad \dfrac{}{f:\forall\emptyset.\tau_2 \vdash f:\tau_5}\,(\text{vars})}{f:\forall\emptyset.\tau_2 \vdash f\,f:\tau_3}\,(\text{app})}{\vdash \lambda f.\,f\,f:\tau_1}\,(\text{abs})$$

(1) $\forall\emptyset\,\tau_2 \succ \tau_4 \Rightarrow \tau_2 = \tau_4 \doteq \tau_5$

(3) $\forall\emptyset\,\tau_2 \succ \tau_5 \Rightarrow \tau_2 = \tau_5$

(3) $\tau_4 = \tau_5 \to \tau_3$

$\tau_2 = \tau_2 \to \tau_3$

can't unify, not equal (for finite types)

Monomorphic types . . .

$$\tau ::= \alpha \mid \mathit{bool} \mid \tau \to \tau \mid \tau\ \mathit{list}$$

. . . and type schemes

$$\sigma ::= \tau \mid \forall\,\alpha\,(\sigma)$$

Polymorphic types

$$\pi ::= \alpha \mid \mathit{bool} \mid \pi \to \pi \mid \pi\ \mathit{list} \mid \forall\,\alpha\,(\pi)$$

E.g. $\alpha \to \alpha'$ is a type, $\forall\,\alpha\,(\alpha \to \alpha')$ is a type scheme and a polymorphic type (but not a monomorphic type), $\forall\,\alpha\,(\alpha) \to \alpha'$ is a polymorphic type, but not a type scheme.

Identity, Generalisation and Specialisation

$$\Gamma \vdash x : \pi \quad \text{if } (x : \pi) \in \Gamma \tag{id}$$

$$\frac{\Gamma \vdash M : \pi}{\Gamma \vdash M : \forall\, \alpha\, (\pi)} \quad \text{if } \alpha \notin \mathit{ftv}(\Gamma) \tag{gen}$$

$$\frac{\Gamma \vdash M : \forall\, \alpha\, (\pi)}{\Gamma \vdash M : \pi[\pi'/\alpha]} \tag{spec}$$

Example

$$
\text{(app)}\ \dfrac{\text{(spec)}\ \dfrac{\text{(gen)}\ \dfrac{\text{(abs)}\ \dfrac{\text{(id)}\ \dfrac{}{x : \alpha \vdash x : \alpha}}{\vdash \lambda x . x : \alpha \to \alpha}}{\vdash \lambda x . x : \forall \alpha . (\alpha \to \alpha)}}{\vdash \lambda x . x : \mathrm{bool} \to \mathrm{bool}} \qquad \text{(true)}\ \dfrac{}{\vdash \mathrm{true} : \mathrm{bool}}}{\vdash (\lambda x . x)\,\mathrm{true} : \mathrm{bool}}
$$

Fact (see Wells (1994)):

$\boxed{\text{For the modified ML type system with polymorphic types and } (\mathrm{var} \succ) \text{ replaced by the axiom and rules on Slide 41, } \textit{the type checking and typeability problems} \text{ (cf. Slide 9) } \textit{are equivalent and undecidable.}}$

Explicitly versus implicitly typed languages

Implicit: little or no type information is included in program phrases and typings have to be inferred (ideally, entirely at compile-time). (E.g. Standard ML.)

Explicit: most, if not all, types for phrases are explicitly part of the syntax. (E.g. Java.)

E.g. self application function of type $\forall\,\alpha\,(\alpha) \to \forall\,\alpha\,(\alpha)$ (cf. Example 7)
Implicitly typed version: $\lambda\,f\,(f\;f)$
Explicitly type version: $\lambda\,f : \forall\,\alpha_1\,(\alpha_1)\,(\Lambda\,\alpha_2\,(f(\alpha_2 \to \alpha_2)(f\;\alpha_2)))$

PLC syntax

Types

$$
\begin{array}{rcll}
\tau & ::= & \alpha & \text{type variable} \\
 & | & \tau \to \tau & \text{function type} \\
 & | & \forall\, \alpha\, (\tau) & \forall\text{-type}
\end{array}
$$

Expressions

$$
\begin{array}{rcll}
M & ::= & x & \text{variable} \\
 & | & \lambda\, x : \tau\, (M) & \text{function abstraction} \\
 & | & M\, M & \text{function application} \\
 & | & \Lambda\, \alpha\, (M) & \text{type generalisation} \\
 & | & M\, \tau & \text{type specialisation}
\end{array}
$$

(α and x range over fixed, countably infinite sets TyVar and Var respectively.)

Functions on types

In PLC, $\boxed{\Lambda\,\alpha\,(M)}$ is an anonymous notation for the function F mapping each type τ to the value of $M[\tau/\alpha]$ (of some particular type).

$\boxed{F\,\tau}$ denotes the result of applying such a function to a type.

Computation in PLC involves beta-reduction for such functions on types

$$(\Lambda\,\alpha\,(M))\,\tau \to M[\tau/\alpha]$$

as well as the usual form of beta-reduction from λ-calculus

$$(\lambda\,x : \tau\,(M_1))\,M_2 \to M_1[M_2/x]$$

PLC typing judgement

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

- the typing environment Γ is a finite function from variables to PLC types.
  (We write $\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$ to indicate that Γ has domain of definition $dom(\Gamma) = \{x_1, \ldots, x_n\}$ and maps each x_i to the PLC type τ_i for $i = 1..n$.)
- M is a PLC expression
- τ is a PLC type.

PLC type system

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

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

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

$$\frac{\Gamma \vdash M : \tau}{\Gamma \vdash \Lambda\, \alpha\,(M) : \forall\, \alpha\,(\tau)} \quad \text{if } \alpha \notin \mathit{ftv}(\Gamma) \qquad \text{(gen)}$$

$$\frac{\Gamma \vdash M : \forall\, \alpha\,(\tau_1)}{\Gamma \vdash M\, \tau_2 : \tau_1[\tau_2/\alpha]} \qquad \text{(spec)}$$

Example (gen/spec)

$$\dfrac{\dfrac{\emptyset \vdash M : \alpha}{\emptyset \vdash \Lambda \alpha\, M : \forall \alpha . \alpha}\ \text{gen}}{\emptyset \vdash (\Lambda \alpha\, M)\, \mathrm{int} : \mathrm{int}}\ \text{spec}$$

Exercise (5 mins)

Consider the identity function *id*, which in the simply-typed lambda calculus is written $\lambda x.x$.

Define *id* in the polymorphic lambda-calculus such that it has type:

$$id : \forall\alpha(\alpha \to \alpha)$$

Give its type derivation tree.

Hint: the polymorphic identity function has two layers of abstraction: first type abstraction over the *type* variable α, then over the *value* variable.

Exercise answer.

(polymorphic identity function)

$$\dfrac{\dfrac{\dfrac{}{x:\alpha \vdash x : \alpha}\,\mathrm{var}}{\vdash \lambda x:\alpha.\,x : \alpha \to \alpha}\,\mathrm{fn}}{\vdash \Lambda\alpha\,\lambda x:\alpha.\,x : \forall\alpha(\alpha \to \alpha)}\,\mathrm{gen}$$

Some syntax considerations

- Application is left associative

$$M_1 M_2 M_3 = (M_1 M_2) M_3$$

- Function type arrows are right associative

$$\tau_1 \to \tau_2 \to \tau_3 \quad = \quad \tau_1 \to (\tau_2 \to \tau_3)$$

- Delimit binders with parentheses; alternatively dot with scope as far to right as possible

$$\forall\alpha.\tau = \forall\alpha(\tau)$$

- Multiple binders

$$\forall\alpha\,(\forall\beta\,(\tau)) = \forall\alpha,\beta\,(\tau)$$
$$\Lambda\alpha\,(\Lambda\beta\,(\tau)) = \Lambda\alpha,\beta\,(\tau)$$

α-equivalence

$$
\begin{aligned}
\Lambda\alpha(\lambda(x:\alpha)x) &= \Lambda\beta(\lambda(x:\beta)x) \\
&= \Lambda\beta(\lambda(y:\beta)y)
\end{aligned}
$$

$$\forall\alpha(\alpha \to \alpha) = \forall\beta(\beta \to \beta)$$

$$
\begin{aligned}
\forall\alpha(\alpha \to \beta \to \alpha) &\neq \forall\beta(\beta \to \beta \to \beta) \\
&\neq \forall\alpha(\alpha \to \gamma \to \alpha)
\end{aligned}
$$

An incorrect 'proof'

$$
\cfrac{
  \cfrac{
    \cfrac{}{x_1 : \alpha, x_2 : \alpha \vdash x_2 : \alpha}\,(\mathrm{var})
  }{x_1 : \alpha \vdash \lambda x_2 : \alpha\,(x_2) : \alpha \to \alpha}\,(\mathrm{fn})
}{x_1 : \alpha \vdash \Lambda\,\alpha\,(\lambda x_2 : \alpha\,(x_2)) : \forall\,\alpha\,(\alpha \to \alpha)}\,(\textbf{wrong}!)
$$

$$\text{(var)}\quad \frac{}{x_1 : \alpha,\ x_2 : \alpha' \vdash x_2 : \alpha'}$$

$$\text{([illegible])}\quad \frac{}{x_1 : \alpha \vdash \underbrace{\lambda x_2 : \alpha'}\ : \alpha' \to \alpha'}$$

$$\text{(gen)}\quad \frac{}{x_1 : \alpha \vdash \Lambda \alpha' (\lambda x_2 : \alpha' (x_2)) : \forall \alpha' (\alpha' \to \alpha')}$$

Explicit types let us control the variables and choose a different (non-conflicting) variable name for the type of x2

Decidability of the PLC typeability and type-checking problems

Theorem.
For each PLC typing problem, $\Gamma \vdash M : ?$, there is at most one PLC type τ for which $\Gamma \vdash M : \tau$ is provable. Moreover there is an algorithm, *typ*, which when given any $\Gamma \vdash M : ?$ as input, returns such a τ if it exists and *FAIL*s otherwise.

Corollary.
The PLC type checking problem is decidable: we can decide whether or not $\Gamma \vdash M : \tau$ is provable by checking whether $\mathit{typ}(\Gamma \vdash M : ?) = \tau$.

(N.B. equality of PLC types up to alpha-conversion is decidable.)

PLC type-checking algorithm, I

Variables:

$typ(\Gamma, x : \tau \vdash x : ?) \stackrel{\text{def}}{=} \tau$

Function abstractions:

$typ(\Gamma \vdash \lambda\, x : \tau_1\,(M) : ?) \stackrel{\text{def}}{=}$
$\text{let } \ \tau_2 = typ(\Gamma, x : \tau_1 \vdash M : ?) \text{ in } \tau_1 \to \tau_2$

Function applications:

$$
\begin{array}{l}
typ(\Gamma \vdash M_1\, M_2 : ?) \stackrel{\text{def}}{=} \\
\ \text{let } \tau_1 = typ(\Gamma \vdash M_1 : ?) \text{ in} \\
\ \text{let } \tau_2 = typ(\Gamma \vdash M_2 : ?) \text{ in} \\
\ \text{case } \tau_1 \ \text{ of } \quad \tau \to \tau' \quad \mapsto \quad \text{if } \tau = \tau_2 \text{ then } \tau' \text{ else } \mathit{FAIL} \\
\qquad\qquad\quad \mid \qquad\quad \_ \quad \mapsto \quad \mathit{FAIL}
\end{array}
$$

PLC type-checking algorithm, II

Type generalisations:

$$
\begin{array}{l}
\mathit{typ}(\Gamma \vdash \Lambda\,\alpha\,(M) : ?) \stackrel{\text{def}}{=} \\
\ \ \text{let } \tau = \mathit{typ}(\Gamma \vdash M : ?) \text{ in } \forall\,\alpha\,(\tau)
\end{array}
$$

Type specialisations:

$$
\begin{array}{l}
\mathit{typ}(\Gamma \vdash M\,\tau_2 : ?) \stackrel{\text{def}}{=} \\
\ \ \text{let } \tau = \mathit{typ}(\Gamma \vdash M : ?) \text{ in} \\
\ \ \text{case } \tau \text{ of } \begin{array}{rcl} \forall\,\alpha\,(\tau_1) & \mapsto & \tau_1[\tau_2/\alpha] \\ \mid \qquad\qquad \_ & \mapsto & \mathit{FAIL} \end{array}
\end{array}
$$

Polymorphic booleans

$$bool \stackrel{\text{def}}{=} \forall\,\alpha\,(\alpha \to (\alpha \to \alpha))$$

$$True \stackrel{\text{def}}{=} \Lambda\,\alpha\,(\lambda\,x_1 : \alpha, x_2 : \alpha\,(x_1))$$

$$False \stackrel{\text{def}}{=} \Lambda\,\alpha\,(\lambda\,x_1 : \alpha, x_2 : \alpha\,(x_2))$$

$$if \stackrel{\text{def}}{=} \Lambda\,\alpha\,(\lambda\,b : bool, x_1 : \alpha, x_2 : \alpha\,(b\,\alpha\,x_1\,x_2))$$