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

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

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

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

$$
\dfrac{
\dfrac{\dfrac{\dfrac{}{x_1:\tau_3,\, x_2:\tau_5 \vdash x_1 : \tau_6}\,\text{var}}{x_1:\tau_3 \vdash \lambda x_2.(x_1) : \tau_4}\,\text{abs}}{\emptyset \vdash \lambda x_1.(\lambda x_2.\, x_1) : \tau_2}\,\text{abs}
\qquad
\dfrac{\dfrac{}{f:\forall A.\tau_2 \vdash f : \tau_7}\,\text{var} \quad \dfrac{}{f:\forall A.\tau_2 \vdash f : \tau_8}\,\text{var}}{f:\forall A.\tau_2. \vdash f\ f : \tau_1}\,\text{app}
}{\emptyset \vdash \text{let } f = \lambda x_1.(\lambda x_2.(x_1)) \text{ in } f\ f : \tau_1}\,\text{let}
$$

Constraints generated while inferring a type for $\texttt{let}\, f = \lambda x_1(\lambda x_2(x_1))\, \texttt{in}\, f\, f$

$$A = ftv(\tau_2) \tag{C0}$$

$$\tau_2 = \tau_3 \to \tau_4 \tag{C1}$$

$$\tau_4 = \tau_5 \to \tau_6 \tag{C2}$$

$$\forall\,\{\,\}\,(\tau_3) \succ \tau_6, \text{ i.e. } \tau_3 = \tau_6 \tag{C3}$$

$$\tau_7 = \tau_8 \to \tau_1 \tag{C4}$$

$$\forall\, A\,(\tau_2) \succ \tau_7 \tag{C5}$$

$$\forall\, A\,(\tau_2) \succ \tau_8 \tag{C6}$$

Constraint solving

$$\tau_2 \overset{(c1)}{=} \tau_3 \to \tau_4 \overset{(c2)}{=} \tau_3 \to (\tau_5 \to \tau_6) \overset{(c3)}{=} \tau_6 \to (\tau_5 \to \tau_6)$$

$$\left.\begin{aligned} \tau_6 &= \alpha_1 \\ \tau_5 &= \alpha_2 \end{aligned}\right\} \text{type variables}$$

$$A = ftv(\tau_2) = \{\alpha_1, \alpha_2\}$$

$$\forall \alpha_1, \alpha_2 . \alpha_1 \to (\alpha_2 \to \alpha_1) \succ \tau_7 \overset{(c4)}{=} \tau_8 \to \tau_1$$

$$\text{"} \qquad\qquad\qquad \text{"} \succ \tau_8$$

(1) $T_8 \to T_1 = T_9 \to (T_{10} \to T_9)$

(2) $T_8 = T_{11} \to (T_{12} \to T_{11})$

$\therefore$ by (1) $T_8 = T_9$

$T_1 = (T_{10} \to T_9)$

$= T_{10} \to (T_{11} \to (T_{12} \to T_{11}))$

Therefore

$$\{\} \vdash \text{let } f = \lambda x_1 . (\lambda x_2 . x_1) \text{ in } f\,f$$

$$: \tau_{10} \to (\tau_{11} \to (\tau_{12} \to \tau_{11}))$$

holds for any $\tau_{10}, \tau_{11}, \tau_{12}$ (they have no more constraints)

$$\therefore \boxed{\begin{array}{l} \{\} \vdash \text{let } f = \lambda x_1 . (\lambda x_2 . x_1) \text{ in } f\,f \\ \quad : \forall \alpha_3, \alpha_4, \alpha_5 . \alpha_3 \to (\alpha_4 \to (\alpha_5 \to \alpha_4)) \end{array}}$$

[p. 17]

The constraints generated from trying to type:

$$(\lambda f\,(f\,f))\;(\lambda x_1.(\lambda x_2.\,x_1))$$

(see slide 23)

gives a constraint

$$\tau_7 \underset{(C13)}{=} \tau_4 \underset{(C12)}{=} \tau_6 \underset{(C11)}{=} \tau_7 \to \tau_5$$

$$\neq$$

Principal type schemes for closed expressions

A closed type scheme $\forall A\,(\tau)$ is the principal type scheme of a closed Mini-ML expression M if

(a) $\vdash M : \forall A\,(\tau)$

(b) for any other closed type scheme $\forall A'\,(\tau')$,
if $\vdash M : \forall A'\,(\tau')$, then $\forall A\,(\tau) \succ \tau'$

Theorem (Hindley; Damas-Milner)

Theorem

If the closed Mini-ML expression M is typeable (i.e. $\vdash M : \sigma$ holds for some type scheme σ), then there is a principal type scheme for M.

Indeed, there is an algorithm which, given any M as input, decides whether or not it is typeable and returns a principal type scheme if it is.

An ML expression with a principal type scheme hundreds of pages long

$$
\begin{array}{l}
\texttt{let}\; pair = \lambda x(\lambda y(\lambda z(z\,x\,y)))\;\texttt{in} \\
\quad \texttt{let}\; x_1 = \lambda y(pair\; y\; y)\;\texttt{in} \\
\quad\quad \texttt{let}\; x_2 = \lambda y(x_1(x_1\, y))\;\texttt{in} \\
\quad\quad\quad \texttt{let}\; x_3 = \lambda y(x_2(x_2\, y))\;\texttt{in} \\
\quad\quad\quad\quad \texttt{let}\; x_4 = \lambda y(x_3(x_3\, y))\;\texttt{in} \\
\quad\quad\quad\quad\quad \texttt{let}\; x_5 = \lambda y(x_4(x_4\, y))\;\texttt{in} \\
\quad\quad\quad\quad\quad\quad x_5(\lambda y(y))
\end{array}
$$

(Taken from Mairson (1990).)

Unification of ML types

There is an algorithm *mgu* which when input two Mini-ML types τ_1 and τ_2 decides whether τ_1 and τ_2 are unifiable, i.e. whether there exists a type-substitution $S \in \mathrm{Sub}$ with

(a) $S(\tau_1) = S(\tau_2)$.

Moreover, if they are unifiable, $mgu(\tau_1, \tau_2)$ returns the most general unifier—an S satisfying both (a) and

(b) for all $S' \in \mathrm{Sub}$, if $S'(\tau_1) = S'(\tau_2)$, then $S' = TS$ for some $T \in \mathrm{Sub}$

(any other substitution S' can be factored through S, by specialising S with T)

By convention $mgu(\tau_1, \tau_2) = \mathit{FAIL}$ if (and only if) τ_1 and τ_2 are not unifiable.

mgu

$$\mathrm{mgu}(T_1, T_2) = S \qquad \mathrm{mgu}(\mathrm{Int}, \mathrm{Bool}) = \mathrm{FAIL}$$

$$T_1 = \alpha \to \mathrm{Bool} \qquad S(T_1) = \beta \to \mathrm{Bool}$$

$$T_2 = \beta \to \gamma \qquad S(T_2) = \beta \to \mathrm{Bool}$$

$$S = [\alpha \mapsto \beta, \ \gamma \mapsto \mathrm{Bool}]$$

"map all variables to Bool"

$$S' = [* \mapsto \mathrm{Bool}] \qquad S'(T_1) = S'(T_2) = \mathrm{Bool} \to \mathrm{Bool}$$

$$S' = [\beta \mapsto \mathrm{Bool}] \circ S$$

Principal type schemes for open expressions

A solution for the typing problem $\Gamma \vdash M : ?$ is a pair $\boxed{(S, \sigma)}$ consisting of a type substitution S and a type scheme σ satisfying

$$S\,\Gamma \vdash M : \sigma$$

(where $S\,\Gamma = \{x_1 : S\,\sigma_1, \ldots, x_n : S\,\sigma_n\}$, if $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$).

Such a solution is principal if given any other, (S', σ'), there is some $T \in \mathrm{Sub}$ with $TS = S'$ and $T(\sigma) \succ \sigma'$.

[For type schemes σ and σ', with $\sigma' = \forall\, A'\,(\tau')$ say, we define $\boxed{\sigma \succ \sigma'}$ to mean $A' \cap \mathit{ftv}(\sigma) = \{\}$ and $\sigma \succ \tau'$.]

Type inference algorithm

$$pt(\emptyset \vdash M : ?) = \tau$$

$$\Downarrow$$

$$pt(\Gamma \vdash M : ?) = (S, \tau)$$

s.t. $S\Gamma \vdash M : \sigma$

where $\sigma = \forall A . \tau$

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

$$\left(S\{x_1 : \tau_1 \ldots x_n : \tau_n\} = x_1 : S\tau_1 \ldots x_n : S\tau_n \right)$$

Example typing problem:

$$\boxed{x : \forall \alpha (\beta \to (\gamma \to \alpha)) \vdash x \text{ true} ?}$$

Has solutions

$S_1 = \{\beta \mapsto \text{bool}\}, \ \sigma_1 = \forall \alpha (\gamma \to \alpha)$

BOTH PRINCIPAL SOLUTIONS

$S_2 = \{\beta \mapsto \text{bool}, \gamma \mapsto \alpha\}, \ \sigma_2 = \forall \alpha' (\alpha \to \alpha')$

$S_3 = \{\beta \mapsto \text{bool}, \gamma \mapsto \alpha\}, \ \sigma_3 = \forall \alpha' (\alpha \to (\alpha' \to \alpha'))$

$S_4 = \{\beta \mapsto \text{bool}, \gamma \mapsto \text{bool}\}, \ \sigma_4 = \forall \{\} (\text{bool} \to \text{bool})$

Properties of the Mini-ML typing relation

- If $\Gamma \vdash M : \sigma$, then for any type substitution $S \in \mathrm{Sub}$
$$S\Gamma \vdash M : S\sigma$$

- If $\Gamma \vdash M : \sigma$ and $\sigma \succ \sigma'$, then $\Gamma \vdash M : \sigma'$.

Specification for the principal typing algorithm, *pt*

pt operates on typing problems $\Gamma \vdash M : ?$ (consisting of a typing environment Γ and a Mini-ML expression M).

It returns either a pair (S, τ) consisting of a type substitution $S \in \mathrm{Sub}$ and a Mini-ML type τ, or the exception *FAIL*.

- If $\Gamma \vdash M : ?$ has a solution (cf. Slide 2), then $pt(\Gamma \vdash M : ?)$ returns (S, τ) for some S and τ;
  moreover, setting $A = (ftv(\tau) - ftv(S\,\Gamma))$, then $(S, \forall\, A\,(\tau))$ is a principal solution for the problem $\Gamma \vdash M : ?$.
- If $\Gamma \vdash M : ?$ has no solution, then $pt(\Gamma \vdash M : ?)$ returns *FAIL*.

Some of the clauses in a definition of *pt*

Function abstractions: $pt(\Gamma \vdash \lambda x(M) : ?) \stackrel{\text{def}}{=}$

$\quad$ let α = fresh in

$\quad$ let $(S, \tau) = pt(\Gamma, x : \alpha \vdash M : ?)$ in $(S, S(\alpha) \to \tau)$

Function applications: $pt(\Gamma \vdash M_1\, M_2 : ?) \stackrel{\text{def}}{=}$

$\quad$ let $(S_1, \tau_1) = pt(\Gamma \vdash M_1 : ?)$ in

$\quad$ let $(S_2, \tau_2) = pt(S_1\, \Gamma \vdash M_2 : ?)$ in

$\quad$ let α = fresh in

$\quad$ let $S_3 = mgu(S_2\, \tau_1, \tau_2 \to \alpha)$ in $(S_3 S_2 S_1, S_3(\alpha))$

Question

$$pt(\{\}, \lambda y.(\lambda x.x+0)y) = ?$$

where forall type variables α

$$pt((x:\alpha, \Gamma), x+0)$$

$$= ([\alpha \mapsto \mathrm{Int}], \mathrm{Int})$$

$pt(\emptyset \vdash \lambda y.((\lambda x. x+0)\, y)) = ?$ $Int \to Int$

abs

$\alpha = fresh$

$(S, \tau) = pt\;(y{:}\alpha \vdash (\lambda x. x+0)\, y)$

app

$(S_1, \tau_1) = pt(y{:}\alpha \vdash \lambda x. x+0)$

abs

$\beta = fresh$

$(S, \tau) = p(x{:}\beta, y{:}\alpha \vdash x+0)$

not defined in the notes
assume this maps B to Int

$= ([\overset{S=}{\beta} \mapsto Int], Int)$

in $(S, S(\beta) \to Int)$

$= (\underset{S_1}{S}, \underset{\tau_1}{\underline{Int \to Int}})$

$(S_2, \tau_2) = pt(S_1(y{:}\alpha) \vdash y)$

var

let $\forall \emptyset\, \tau = \alpha$

in $(\underset{S_2}{Id}, \underset{\tau_2}{\alpha})$

$\gamma = fresh$

$S_3 = mgu(S_2\tau_1, \tau_2 \to \gamma)$

$mgu(Int \to Int, \alpha \to \gamma)$

$= [\alpha \mapsto Int, \gamma \mapsto Int]$

in $([\alpha \mapsto Int, \gamma \mapsto Int, \beta \mapsto Int], S_3\gamma)$

$= Int$

$S = [\alpha, \gamma, \beta \mapsto Int, Int]$

in $(S, S(\alpha) \to Int$

$(S, Int \to Int)$