Polymorphism The major new ingredient is type variables within the type system. We use letters like $\alpha, \beta, \gamma \ldots$ as type variables. Our language of types is now

$$\tau ::= \text{int} \mid \text{bool} \mid \ldots \mid \tau_1 * \tau_2 \mid \tau_1 \to \tau_2 \mid \alpha$$

What does it mean to say an expression e has type $\alpha * \alpha$? It means that any type expression substituted consistently for α gives a possible type of e. Thus it means that e belongs to any one of a family of types.

What are these substitutions? We define by induction the notation $[\tau/\alpha]\tau'$: replace occurrences of α in τ' by τ.

(1) $[\tau/\alpha]\alpha = \tau$

(2) $[\tau/\alpha]\beta = \beta \quad (\alpha \neq \beta)$

(3) $[\tau/\alpha]\text{int} = \text{int}$

(4) $[\tau/\alpha]\text{bool} = \text{bool}$

(5) $[\tau/\alpha]\tau_1 * \tau_2 = ([\tau/\alpha]\tau_1) * ([\tau/\alpha]\tau_2)$

(6) $[\tau/\alpha]\tau_1 \to \tau_2 = ([\tau/\alpha]\tau_1) \to ([\tau/\alpha]\tau_2)$.

Here is the crucial rule that captures polymorphism

If $\Gamma \vdash e : \tau$ then $[\tau'/\alpha]\Gamma \vdash e : [\tau'/\alpha]\tau$.

In our definition of substitution there is nothing that says τ cannot have its own type variables.

Consider $\quad$ fun $x \to x$

It can be given many monotypes

$$\frac{x : \text{int} \vdash x : \text{int}}{\vdash \text{fun } x \to x : \text{int} \to \text{int}}$$

or

$$\frac{x : \text{int} \to \text{int} \vdash x : \text{int} \to \text{int}}{\vdash \text{fun } x \to x : (\text{int} \to \text{int}) \to (\text{int} \to \text{int})}$$

These types for fun $x \to x$ are all obtained by making appropriate substitutions to $\alpha \to \alpha$.
We call them substitution instances of $\alpha \to \alpha$.

Thm For every expression e, there is a unique type τ, possibly containing type variables such that every valid type for e is obtained by an appropriate substitution of τ. We say that τ is a (or the) principal type for e.

TYPE INFERENCE

The strategy is to introduce fresh type variables whenever we don't know the type of a sub-expression. Then we look at how the expressions are used to infer constraints on the type variables. In the final phase, we try to solve the constraints. We will look for the most general solution: this means that we are looking for a solution such that all other possible solutions are substitution instances of the most general one.

NOTATION $\Gamma \vdash e : \tau / C$

in the context Γ, the expression e will have type τ if the constraints in C are satisfied.
The constraints are of the form $\tau_1 = \tau_2$.
For constants we do not generate constraints.

$$\frac{}{\Gamma \vdash n : int \,/\, \phi} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau \,/\, \phi}$$

$$\frac{\Gamma \vdash e : bool/C_0 \qquad \Gamma \vdash e_1 : \tau_1/C_1 \qquad \Gamma \vdash e_2 : \tau_2/C_2}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau \,/\, C_0 \cup C_1 \cup C_2 \cup \{\tau_1 = \tau_2\}}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 / C_1 \qquad \Gamma \vdash e_2 : \tau_2 / C_2}{\Gamma \vdash e_1 + e_2 : int \,/\, C_1 \cup C_2 \cup \{\tau_1 = int, \tau_2 = int\}}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 / C_1 \qquad \Gamma \vdash e_2 : \tau_2 / C_2}{\Gamma \vdash e_1 = e_2 : bool \,/\, C_1 \cup C_2 \cup \{\tau_1 = \tau_2\}}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 / C_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2 / C_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2 / C_1 \cup C_2}$$

<u>Functions</u> We don't have declarations so how can we know the type of x in fun $x \to \ldots$? We don't know, so we introduce a <u>fresh</u> type variable say α:

$$\frac{\Gamma, x : \alpha \vdash e : \tau / C}{\Gamma \vdash \text{fun } x \to e : \alpha \to \tau / C}$$

<u>Example</u>

$$\cfrac{\cfrac{\cfrac{}{x : \alpha \vdash x : \alpha / \phi} \qquad \cfrac{}{x : \alpha \vdash 1 : int / \phi}}{x : \alpha \vdash x + 1 : int \,/\, \{\alpha = int\}}}{\vdash \text{fun } x \to x+1 : \alpha \to int \,/\, \{\alpha = int\}}$$

solution $\alpha = int$ so we get

$$\frac{}{\vdash \text{fun } x \to x+1 : int \to int}$$

Applications: We have to guess the return type of $e_1 e_2$ by introducing a fresh type variable:

$$\frac{\Gamma \vdash e_1 : \tau_1 / C_1 \qquad \Gamma \vdash e_2 : \tau_2 / C_2}{\Gamma \vdash e_1 e_2 : \alpha \,/\, C_1 \cup C_2 \cup \{\tau_1 = \tau_2 \to \alpha\}}$$

Lists $$\frac{\Gamma \vdash e_1 : \tau_1 / C_1 \qquad \Gamma \vdash e_2 : \tau_2 / C_2}{\Gamma \vdash (e_1 :: e_2) : \tau_2 \,/\, C_1 \cup C_2 \cup \{\tau_2 = \tau_1\text{-list}\}}$$

$$\frac{}{\vdash [\,] : \alpha\text{-list}} \qquad \frac{\Gamma \vdash l : \tau\text{-list} / C}{\Gamma \vdash head(l) : \tau / C} \qquad \frac{\Gamma \vdash l : \tau\text{-list} / C}{\Gamma \vdash tail(l) : \tau\text{-list} / C}$$

EXAMPLES OF INFORMAL DERIVATIONS

let rec map = fun f -> fun x -> if (x = []) then []
else f(head(x)) :: (map f (tail(x))).

We introduce variables for the types that we do not ~~see~~ know and then look for constraints:

$f : \alpha$, $x : \beta$

From $x = [\,]$ we see $\beta = \gamma$-list

From $f(head(x))$ we see f is a function and $head(x) : \gamma$ so $\alpha = \gamma \to \delta$ δ is fresh.

$f(head(x)) = \delta$ so $f(head(x)) :: \cdots : \delta$-list.

So type of map is

$$(\gamma \to \delta) \to \gamma\text{-list} \to \delta\text{-list}.$$

```
let rec append (l1, l2) =
    match l1 with
      | [] -> l2
      | x::xs -> x:: (append(xs, l2)).
```

$l_1 : \alpha \quad l_2 : \beta$

from the match : $\alpha = \gamma$- list $\quad \gamma$- fresh

$x : \gamma$ so return type is γ- list

so ~~f~~ $l_2 : \gamma$- list

Thus $\quad \gamma$- list $*$ γ- list $\rightarrow \gamma$- list.

let double = fun f → fun x → f (f x)

$f : \alpha, \quad x : \beta \qquad f\ x : \gamma$

so $f : \cancel{\alpha \rightarrow} \beta \rightarrow \gamma = \alpha$

f (f x) says input type for f is γ

so $\beta = \gamma$

double : $(\beta \rightarrow \beta) \rightarrow \beta \rightarrow \beta$

~~let~~ fun x → fun y → (x, y)

$\alpha \rightarrow \beta \rightarrow \alpha * \beta$

fun f → f f

$f : \alpha \qquad$ f f says $f : \alpha \rightarrow \beta$

so $\boxed{\alpha = \alpha \rightarrow \beta}$

This equation cannot be solved!