Monomorphic, explicitly-typed OCaml

```
type type = IntType | BoolType | FunType of type * type

type exp = Int of int | True | False | Var of string
         | App of exp * exp | Fun of string * type * exp
         | Operation of exp * binary_operation * exp
```

Type rules (where Γ is a mapping from variables to types, and each binary operation $\oplus$ is assumed to have a given type $\tau \longrightarrow \tau' \longrightarrow \tau''$):

(Const)	$\Gamma \vdash \textbf{Int i} : \textbf{int}$	**(Var)**	$\Gamma \vdash a : \Gamma(a)$
(Fun)	$\Gamma \vdash \textbf{fun } a{:}\tau \texttt{ -> } e : \tau \longrightarrow \tau'$ $\Gamma[a{:}\tau] \vdash e : \tau'$	**(δ)**	$\Gamma \vdash e \oplus e' : \tau''$ $\Gamma \vdash e : \tau$ $\Gamma \vdash e' : \tau'$
(App)	$\Gamma \vdash e\ e' : \tau'$ $\Gamma \vdash e : \tau \longrightarrow \tau'$ $\Gamma \vdash e' : \tau$	**(True)** **(False)**	$\Gamma \vdash \textbf{true} : \textbf{bool}$ $\Gamma \vdash \textbf{false} : \textbf{bool}$

Examples

$\emptyset \vdash$ `fun x:int -> x+1` : **int** $\longrightarrow$ **int** (Fun)

$\{x:\text{int}\} \vdash x+1 : \text{int}$ (δ)

$\{x:\text{int}\} \vdash x : \text{int}$ (Var)

$\{x:\text{int}\} \vdash 1 : \text{int}$ (Const)

$\emptyset \vdash$ `fun f:(int->int) -> f 3` : **(int** $\longrightarrow$ **int)** $\longrightarrow$ **int** (Fun)

$\{f:\text{int}\to\text{int}\} \vdash f\ 3 : \text{int}$ (App)

$\{f:\text{int}\to\text{int}\} \vdash f : \text{int}\to\text{int}$ (Var)

$\{f:\text{int}\to\text{int}\} \vdash 3 : \text{int}$ (Const)

Examples (cont.)

$\emptyset \vdash$ `fun f:(int->int) -> fun x:int -> f (f x)` (Fun)

$: (\text{int} \rightarrow \text{int}) \rightarrow (\text{int} \rightarrow \text{int})$

$\{f : int \to int\} \vdash fun\ x : int \to f\ (f\ x) : int \to int$ (Fun)

Γ $\{f : int \to int, x : int\} \vdash f\ (f\ x) : int$ (App)

$\Gamma \vdash f : int \to int$ (Var)

$\Gamma \vdash f\ x : int$ (App)

$\Gamma \vdash f : int \to int$ (Var)

$\Gamma \vdash x : int$ (Var)

Informal example of type inference

$\emptyset \vdash (\texttt{fun } \texttt{x}_\alpha \texttt{ -> } (\texttt{x}_\alpha\texttt{+1})_{int})_\beta$

$\alpha = int$
$\beta = \alpha \to int$
$\therefore \beta = int \to int$

$\emptyset \vdash (\texttt{fun } \texttt{f}_\alpha \texttt{ -> } (\texttt{1 + } (\texttt{f}_\alpha \texttt{ 3})_\beta)_\gamma)_\epsilon$

$\gamma = int$
$\alpha = int \to \beta$
$\epsilon = \alpha \to \gamma$
$\beta = int$
$\therefore \gamma = int \to int$

$\emptyset \vdash (\texttt{fun } \texttt{f}_\alpha \texttt{ -> } (\texttt{f}_\alpha \texttt{ 3}_\beta)_\gamma)_\epsilon$

$\beta = int$
$\alpha = \beta \to \gamma$
$\epsilon = \alpha \to \gamma$
$\therefore \epsilon = (int \to \gamma) \to \gamma$

Type inference, formally

- Generate a set E of equations, or *constraints*, of the form $\alpha = \tau$, as follows: Start with $E = \emptyset$. For every subexpression t' of t, add constraints to E, depending upon the form of t':

$(\textbf{Int } i)_\alpha$:	$\{\ \alpha = \textbf{int}\ \}$
$\textbf{true}_\alpha$ **or** $\textbf{false}_\alpha$:	$\{\ \alpha = \textbf{bool}\ \}$
$(e_\alpha \oplus e'_\beta)_\gamma$:	$\{\ \alpha = \tau,\ \beta = \tau',\ \gamma = \tau''\ \}$
	(where $\oplus$ **has type** $\tau \rightarrow \tau' \rightarrow \tau''$**)**
$(\textbf{fun } x_\alpha \rightarrow e_\beta)_\gamma$:	$\{\ \gamma = \alpha \rightarrow \beta\ \}$
$(e_\alpha\, e'_\beta)_\gamma$:	$\{\ \alpha = \beta \rightarrow \gamma\ \}$

Example of generating E

$\emptyset \vdash$ `fun f -> f (f 3)`

Annotated: $(\texttt{fun}\ f_\alpha \rightarrow (f_\alpha\ (f_\alpha\ 3_\beta)_\gamma)_\delta)_\psi$

Constraints:

$\beta = int$

$\alpha = \beta \to \gamma$

$\alpha = \gamma \to \delta$

$\psi = \alpha \to \delta$

Solving E (informally)

- **Start from the constraints generated in the previous example:**

$$
\begin{array}{l} \beta = int \\ \alpha = \beta \to \gamma \\ \alpha = \gamma \to \delta \\ \psi = \alpha \to \delta \end{array}
\overset{\beta}{\Rightarrow}
\begin{array}{l} \beta = int \\ \alpha = int \to \gamma \\ \alpha = \gamma \to \delta \\ \psi = \alpha \to \delta \end{array}
\Rightarrow
\begin{array}{l} \beta = int \\ \alpha = int \to int \\ \psi = \alpha \to \delta \\ \gamma = int \\ \delta = int \end{array}
\overset{\alpha}{\Rightarrow}
\begin{array}{l} \beta = int \\ \alpha = int \to int \\ \psi = (int \to int) \to \delta \\ \gamma = int \\ \delta = int \end{array}
\overset{\delta}{\Rightarrow}
\begin{array}{l} \beta = int \\ \alpha = int \to int \\ \psi = (int \to int) \to int \\ \gamma = int \\ \delta = int \end{array}
$$

unify $int \to \gamma$
with $\gamma \to \delta$

- ***(Note how we had to make two type expressions match up — this is called* unification*.)***

Unification examples

- **Will present algorithm for unification later. These are examples to be solved by inspection.**

unify($\alpha \rightarrow \beta$**, int**$\rightarrow \gamma$ **)**

$\alpha \mapsto int$ $\quad$ or $\quad$ $\alpha \mapsto int$

$\beta \mapsto \gamma$ $\quad\quad\quad\;\;$ $\gamma \mapsto \beta$

unify($\alpha \rightarrow$**(int**$\rightarrow \beta$), **(int**$\rightarrow$**int)**$\rightarrow \gamma$ **)**

$\alpha \mapsto int \rightarrow int$

$\gamma \mapsto int \rightarrow \beta$