Last time: Type-directed coding

Common idea in functional programming: “lifting”

```ocaml
val lift : forall 'a . ('a -> bool) -> ('a list -> bool)

fun lift p [] = false
    | lift p (z::zs) = p z orelse lift p zs
```
## Types and their C constructs

<table>
<thead>
<tr>
<th>Type</th>
<th>Produce</th>
<th>Consume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduce</td>
<td></td>
<td>Eliminate</td>
</tr>
</tbody>
</table>

- **struct**: "initializer"  
  ```
  struct {
      ...,
      ...
  }
  ```

- **pointer**: `&`  
  `*`

- **function**: (definition form)  
  `application`

- **dot notation**: `e.next`, `e->next`
## Types and their ML constructs

<table>
<thead>
<tr>
<th>Type</th>
<th>Produce</th>
<th>Consume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduce</td>
<td>Lambda ((\text{fn}))</td>
<td>Application</td>
</tr>
<tr>
<td>Eliminate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arrow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>constructed (algebraic)</td>
<td>Apply constructor</td>
<td>Pattern match</td>
</tr>
<tr>
<td>constructed (tuple)</td>
<td>((e_1, \ldots, e_n))</td>
<td>Pattern match!</td>
</tr>
</tbody>
</table>
Type this: Language of expressions

Numbers and Booleans:

datatype exp = ARITH of arithop * exp * exp
    | CMP of relop * exp * exp
    | LIT of int
    | IF of exp * exp * exp
and arithop = PLUS | MINUS | TIMES | ...
and relop = EQ | NE | LT | LE | GT | GE

datatype ty = INTTY | BOOLTY

Problem to solve: integer register or flags register?
Type checking in ML (no variables!)

val typeof : exp -> ty
exception IllTyped

fun typeof (ARITH (_, e1, e2)) =
  (case (typeof e1, typeof e2)
   of (INTTY, INTTY) => INTTY
    | _ => raise IllTyped)

| typeof (CMP (_, e1, e2)) =
  (case (typeof e1, typeof e2)
   of (INTTY, INTTY) => BOOLTY
    | _ => raise IllTyped)

| typeof (LIT _) = INTTY

| typeof (IF (e,e1,e2)) =
  (case (typeof e, typeof e1, typeof e2)
   of (BOOLTY, tau1, tau2) =>
      if eqType (tau1, tau2)
      then tau1 else raise IllTyped
    | _ => raise IllTyped)