Type-directed coding

Common idea in functional programming: “lifting”

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

What (sensible) functions have this type?
Working...
Type-directed coding (results)

```
val lift : ('a -> bool) -> ('a list -> bool)
fun lift p = (fn xs => (case xs
    of [] => false
    | z::zs => p z orelse
    lift p zs))
```
fun lift p xs = case xs of [] => false
              | z::zs => p z orelse
                 lift p zs
Merge top-level case into fun

fun lift p [] = false
  | lift p (z::zs) = p z orelse lift p zs
I know this function!

fun exists p []    = false
           | exists p (z::zs) = p z orelse exists p zs
### Types and their ML constructs

<table>
<thead>
<tr>
<th>Type</th>
<th>Produce</th>
<th>Consume</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrow</td>
<td>Lambda (fn)</td>
<td>Application</td>
</tr>
<tr>
<td>constructed</td>
<td>Apply constructor</td>
<td>Pattern match</td>
</tr>
<tr>
<td>(algebraic)</td>
<td></td>
<td>Pattern match!</td>
</tr>
<tr>
<td>constructed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(tuple)</td>
<td>$\left(e_1, \ldots, e_n\right)$</td>
<td></td>
</tr>
</tbody>
</table>

- **Types and ML Constructs**
  - **Types**: Introduce, Eliminate
  - **Produce**: Lambda (fn)
  - **Consume**: Application, Pattern match
  - **Arrow**: Apply constructor
  - **Construct** (tuple): $\left(e_1, \ldots, e_n\right)$
Type this: Language of expressions

**Numbers and Booleans:**

```ml
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
```