Lecture 3
September 5
Stephen Brookes
Dilsun Kaynar
Today

• A brief remark about equality types
• Using patterns
• Specifying what a function does
equality in ML

- Infix operator \( e_1 = e_2 \)
- Only for expressions whose type is an equality type
- Equality types include all types built from \( \text{int, bool, *, -list} \) but NOT \( \text{real or -} \to \)

```
- 1+1 = 2;
val it = true : bool
- [1,1] = (0+1)::[2-1];
val it = true : bool
- (fn x => x+x) = (fn y => 2*y);
Error: operator and operand don't agree [equality type required]
```
patterns

- We introduced patterns, to be used for matching with values

- Matching \( p \) to value \( v \) either fails, or succeeds and binds names to values

\[
p ::= \ _ | x | n | \text{true} | \text{false} \\
    | (p_1, \ldots, p_n) \\
    | p_1 :: p_2 \\
    | [p_1, \ldots, p_n]
\]

Can attach types if desired
Constant patterns can only be used to match values of an equality type.
Using patterns

Recall... \( \text{divmod} : \text{int} \times \text{int} \rightarrow \text{int} \times \text{int} \)

```ml
fun check (x:int, y:int):bool = let val (q, r) = divmod (x, y) in (x = q*y + r) end
```

Introduces \( \text{check} : \text{int} \times \text{int} \rightarrow \text{bool} \)

Binds \( \text{check} \) to a function value

**What does this function do?**
**eval : int list -> int**

```haskell
fun eval ([ ]) = 0
    | eval (d::L) = d + 10 * (eval L)
```

This definition uses *list patterns*

- `[ ]` matches (only) the empty list
- `d::L` matches a non-empty list, binds `d` to head of the list, `L` to its tail

**eval [2,4] =>* 42**

What does this function do?
fun decimal n = if n < 10 then [n] else (n mod 10) :: decimal (n div 10)

Why didn’t I define this function using integer patterns?

• decimal 42 = [2,4]
• decimal 0 = [0]

What does this function do?
**log : int -> int**

```
fun log x =
  if x = 1 then 0 else 1 + log (x div 2)
```

- For what argument values does it terminate?
- How does the output relate to the input?

**Q:** How can we *describe* this function?

**A:** *Specify* its *applicative behavior*…

- For what argument values does it terminate?
- How does the output relate to the input?
For each function definition we specify:

- **Type**
  (of the function’s argument and result)

- **Assumption**
  (about argument value)

- **Guarantee**
  (about result value, when assumption holds)
fun log (x:int) : int = 
  if x=1 then 0 else 1 + log (x div 2)

(* TYPE              log : int -> int *)
(* REQUIRES          ... x ... *)
(* ENSURES           ... log x .... *)

For all values x : int satisfying the assumption,
log x : int and satisfies the guarantee

Any ideas?
fun log (x:int) : int = 
  if x=1 then 0 else 1 + log (x div 2)

(* TYPE log : int -> int *)

(* REQUIRES x > 0 *)

(* ENSURES log x = the integer k ≥ 0 *)
  such that 2^k ≤ x < 2^{k+1}

For all integer values x>0, log x evaluates to an integer k such that 2^k ≤ x < 2^{k+1}
notes

- Can use $\Rightarrow^*$ or $=*$ in specs
- Use *math notation* and *math facts*, but do so *accurately*!
- One function can have several specs…

更强的假设
可能保证更强的结果
another log spec

fun log (x:int) : int =
  if x=1 then 0 else 1 + log (x div 2)

(* log : int -> int *)

(* REQUIRES  x = a power of 2 *)

(* ENSURES  log x = the integer k *)
(*  such that $2^k = x$ *)

(makes stronger assumption
and gives more precise guarantee)
**eval spec**

**fun eval ([ ] : int list) : int = 0**

| eval (d::L) | = d + 10 * (eval L) |

**TYPE**

eval : int list -> int

**REQUIRES**

R = a list of decimal digits

**ENSURES**

eval R = a non-negative integer

(not the *best* spec for eval… why not?)

(doesn’t say *which* non-negative integer!)
fun decimal (n:int) : int list = 
  if n<10 then [n] 
  else (n mod 10) :: decimal (n div 10)

TYPE decimal : int -> int list
REQUIRES n ≥ 0
ENSURES decimal n = a list of decimal digits

(again, not the best spec….)
connection

• `eval` and `decimal` are designed to fit together

• They satisfy a **connection spec**

<table>
<thead>
<tr>
<th>TYPE</th>
<th>decimal : int -&gt; int list</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eval : int list -&gt; int</td>
</tr>
</tbody>
</table>

**REQUIRES**  \( n \geq 0 \)

**ENSURES** \( \text{eval(decimal n)} = n \)

(implies the earlier specs!)

(says “which list” and “which integer”)

---

Connection

- `eval` and `decimal` are designed to fit together.
- They satisfy a *connection spec*:
  
  - **TYPE**
    - `decimal : int -> int list`
    - `eval : int list -> int`
  
  - **REQUIRES** \( n \geq 0 \)
  
  - **ENSURES** \( \text{eval(decimal n)} = n \)

(implies the earlier specs!)

(says “which list” and “which integer”)

---