Formal Specification F28FS2, Lecture 15
Operations in ML, especially those on lists

Jamie Gabbay

March 23, 2014

Playing games with ML types

Often, you can deduce what a 'reasonable' function must do, just by looking at its type.

Try this with the type `('a -> 'b) -> 'a list -> 'a list`?

So this is a function that takes two arguments: a function from $\alpha \rightarrow \beta$ and a list of αs.

What could such a function do? Well, there is only one possibility:

```
fun map f [] = []
  | map f (hd::tl) = (f hd)::(map f tl);
val map = fn :  ('a -> 'b) -> 'a list -> 'b list
```

You need to get used to parsing these things.

Another example

Consider this type: `(('a * 'b) -> 'b) -> 'b -> 'a list -> 'b`.

This is a function that takes a function from $\alpha \times \beta$ to β, and a β, and a list of αs, and returns a β.

What could such a function do? Again, there is an obvious possibility.

Try to work it out first, then look at the next slide.

Another example

```
fun foldl f b [] = b
  | foldl f b (hd::tl) = (foldl f (f(hd,b)) tl);
val foldl = fn :  (('a * 'b) -> 'b) -> 'b -> 'a list
-> 'b
```

We can use `foldl` to write an iterative function over a list, such as this:

- Sum: `fn l => foldl (fn (x,y) => x+y) 0 l;`
- Sum squares: `fn l => foldl (fn (x,y) => x*x+y) 0 l;`
- Sum squares (using `map`): `fn l => foldl (fn (x,y) => x+y) 0 (map (fn x => x*x) l);`

More examples

How about `int -> 'a list -> 'a`?

Seems to me this has to be a program that chooses the *n*th element of *l*. Try to write this yourself.

More examples

```
fun take 1 (hd::tl) = hd
  | take n (hd::tl) = take (n-1) tl;
```

Of course this is a partial function. Do we care? Well if we do we can use an exception:

```
exception IndexOutOfBounds;
fun take 1 (hd::tl) = hd
  | take n (hd::tl) = take (n-1) tl
  | take n [] = raise IndexOutOfBounds;
val take = fn :  int -> 'a list -> 'a
```

We get this: `- take 1 [1];`

```
val it = 1 :  int
- take  1 [1];
uncaught exception IndexOutOfBounds
```

Max

Write as many functions as you can to calculate the maximum of a list of integers. The type should be `int list -> int`.

Max

Here are two of mine:

```
fun max (hd::tl) = if hd>(max tl) then hd else (max
tl);
fun max (hd::tl) = fn tl => foldl (fn (x,y) => if x>y
then x else y) hd tl;
```

Of course we can write more elaborate programs that gracefully handle `max` of the empty list. Have a go.

Filter

How about a program of type `('a -> bool) -> 'a list -> 'a list`?

Clearly this is `filter`:

```
fun filter P [] = []
  | filter P (hd::tl) = if (P hd) then hd::(filter P
tl) else (filter P tl);
```

Try writing a function that inputs a list of predicates (a list of functions in $\alpha \to \mathit{bool}$) and a list of αs and outputs the sublist of elements satisfying all of these predicates. So the type should be `('a -> bool) list -> 'a list -> 'a list`.

Exercises

- Write the obvious polymorphic function of type `'a -> int`.
- Recall that in Z, relations $A \leftrightarrow B$ can be modelled as sets of tuples $\mathbb{P}(A \times B)$. As discussed in previous lectures, this has two models in ML: `(A*B) list` and `(A*B) -> bool`. The first is an equality type if `A` and `B` are, the second is not an equality type but can contain infinite elements.
  Recall that predicates on A are modelled as `A -> bool`, and similarly for B.
  For the first model, `(A*B) list`, implement the functions size of relation (the length of the list), domain, range, domain restriction, and range anti-restriction, and state their types.