NEXT · UP · PREVIOUS · CONTENTS · INDEX

Curried functions

The other question which arises once we have discovered that functions may take functions as arguments is ``Can functions return functions as results?'' (Such functions are called curried functions after Haskell B. Curry.) Curried functions seem perfectly reasonable tools for the functional programmer to request and we can encode any curried function using just the subset of Standard ML already introduced, e.g. **fn** x => **fn** y => x.

This tempting ability to define curried functions might leave us with the difficulty of deciding if a new function we wish to write should be expressed in its curried form or take a tuple as an argument. Perhaps we might decide that every function should be written in its fully curried form but this decision has the unfortunate consequence that functions which return tuples as results are sidelined. However, the decision is not really so weighty since we may define functions to curry or uncurry a function after the fact. We will define these after some simple examples.

A simple example of a function which returns a function as a result is a function which, when given a function *f*, returns the function which applies *f* to an argument and then applies *f* to the result. Here is the Standard ML implementation of the twice function.

```
val twice = fn f => fn x => f(f x);
```

For idempotent functions, twice simply acts as the identity function. The integer successor function **fn** x => x+1 can be used as an argument to twice.

```
val addtwo = twice (fn x => x+1);
```

The function twice is only a special case of a more general function which applies its function argument a number of times. We will define the iter function for iteration. It is a curried function which returns a curried function as its result. The function will have the property that iter 2 is twice. In the simplest case we have f^0 = id, the identity function. When *n* is positive we have that $f^n(x) = f(f^{n-1}(x))$. We may now implement the iter function in Standard ML.

```
val rec iter =
  fn 0 => (fn f => fn x => x)
   | n => (fn f => fn x => f (iter (n-1) f x));
```

As promised above, we now define two higher-order, curried Standard ML functions which, respectively, transform a function into its curried form and transform a curried function into tuple-form.

```
val curry = fn f => fn x => fn y => f(x, y);
val uncurry = fn f => fn (x, y) => f x y;
```

If x and y are values and f and g are functions then we always have:

$$(\text{curry } f)\ x\ y = f(x, y)$$

$$(\text{uncurry } g)(x, y) = g\ x\ y$$

NEXT · UP · PREVIOUS · CONTENTS · INDEX