

1998 Paper 13 Question 10

Introduction to Functional Programming

The following datatypes are meant to be used to represent programs written in a simple imperative language:

```
datatype expression = Expr of (string -> int) -> int;
datatype command = Assign of (string * expression)
                 | Sequence of (command * command)
                 | While_do of (expression * command);
```

The *state*, namely the values of the variables at a given point, is represented by a function that takes the variable name and gives the corresponding value. Variables and expressions only involve integers and, when treated as booleans, zero is regarded as *false* and non-zero values as *true*. For example, the compound command:

```
x := 1;
while n <> 0 do
  { x := x * n;
    n := n - 1 }
```

can be written

```
Sequence(Assign("x",Expr(fn s => 1)),
         While_do(Expr(fn s => s"n"),
                  Sequence(Assign("x",Expr(fn s => s"x" * s"n")),
                            Assign("n",Expr(fn s => s"n" - 1)))));
```

A command can be interpreted as a mapping from the initial state to the final state; this will be achieved by the function `interpret` below.

First write an ML function `update` whose type is

```
(string -> int) * string * int -> (string -> int)
```

such that `update(s,x,i)` gives a new state representing state `s` but with the variable `x` being assigned the value `i`. [5 marks]

Now, using `update` or otherwise, write an ML function `interpret` whose type is

```
command -> (string -> int) -> (string -> int)
```

which takes a command `c` and an initial state s_1 , and returns the corresponding final state s_2 that results from executing `c` in state s_1 . For example, if the above compound command is bound to `fp`, then

```
interpret fp (fn "n" => 6) "x";
```

should yield 720.

[15 marks]