End-semester Examination

Abstractions and Paradigms in Programming

Date: 23rd April 2010 Weightage: 50 %
Total Marks: 90 Closed Book Exam

Short Answers

1. Run-length encoding (RLE) is a very simple form of data compression in which runs of data (that is, sequences of the same data value) are stored as a single data value and count. As an example:

   ```
   (rle ‘(1 1 1 3 3 2 5 5 5 4 4)) => ((1 3) (3 2) (2 1) (5 3) (4 2))
   ```

   Define `rle`. (5 Marks)

2. Write a function `(where-smallest f a b)` which takes a function `f` and the integer range `[a, b]` and determines the integer point in the given range where the function has the smallest value. (5 Marks)

3. Prove for the function defined below: $\forall b \forall n.$`(expt b n)` $= b^n$. What auxilliary result do you have to prove about `expt-iter`? (10 Marks).

   ```
   (define (expt b n)
      (expt-iter b n 1))

   (define (expt-iter b counter product)
      (if (= counter 0) product
         (expt-iter b (- counter 1) (* b product))))
   ```

4. Write a procedure `make-monitored` that takes as input a procedure, `f`, that itself takes *one input*. The result returned by `make-monitored` is a third procedure, say `mf`, that keeps track of the number of times `mf` has been called by maintaining an internal counter. If the input to `mf` is the special symbol `how-many-calls?`, then `mf` returns the value of the counter. If the input is the special symbol `reset-count`, then `mf` resets the counter to zero. For any other input, `mf` returns the result of calling `f` on that input and incrementing the counter. As an example: (8 Marks)

   ```
   >(define mf  (make-monitored sqrt))
   >(mf 100)
   10
   >(mf 'how-many-calls?)
   1
   ```

5. Suppose we wanted to represent whole numbers as 0, s(0), s(s(0)), . . . standing for 0, 1, 2 . . . . Write prolog rules for a predicate mult(X,Y,Z) to mean "Z is the product of X and Y". Define additional rules, if required. (7 Marks).

Longer Answers

1. Using the environmental model of execution, clearly explain the outputs due to the `display` statement marked #. (8 Marks)

```
(define (recurse i q)
  (define (p) (# display i))
  (if (> i 0) (recurse (- i 1) p)
      (begin
        (p)
        (q))))
(define (dummy) (display ""))
(define (main) (recurse 1 dummy))
(main)
```

2. We shall represent a binary list (a list of 0's and 1's) using a binary tree in the following way. If the list consists of all 1's or 0's, it is represented as a leaf node with the value 1 or 0. Otherwise the list is divided into two equal sublists and represented as a tree with each sub-tree being a representation of a sublist. Assume that the length of all the lists is the same, and this length, `len`, is some power of 2.

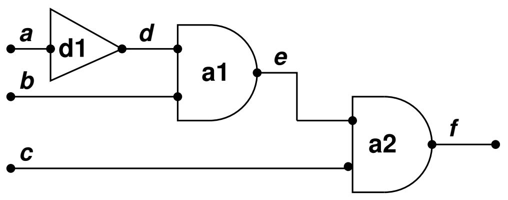
   (a) Define appropriate structures to represent binary lists.

   (b) Define a function called `tobintree` which will take a binary list and convert it into a binary tree.

   (c) Define functions `bitwise-or` and `bitwise-and` which will take the binary tree representation of two binary lists and return a binary tree representing the result of the corresponding operation performed bit-wise.

   (d) Finally, define a function `(value?  i bt)` which will take as input an integer `i` representing an index in a binary list and a binary tree representation of the list, `bt`, and determine whether the value at the index is 0 or 1. (16 Marks)

3. Consider the problem of simulating digital circuits made up of and gates and delays. An example of such a circuit is shown below.



To create the circuit, you would have to say:

```
(define a1 (make-connector))
(define b1 (make-connector))
...
(define f1 (make-connector))

(define del (delay a1 d1))
(define and1 (and-gate d1 b1 e1))
(define and2 (and-gate e1 c1 f1))
```

There is a scheduler which is passed two lists, a list of all connectors (in any order) and a list of all gates (in any order).

```
(define sch (scheduler (list a1 b1 c1 d1 e1 f1) (list d1 a1 a2)))
```

Initially all connectors have the value `'undefined`. The circuit is given input as follows:

```
> (set-value! a1 #t)
> (set-value! b1 #f)
> (set-value! c1 #t)

> (sch 'tick)
a1 #t, b1 #f, c1 #t, d1 undefined, e1 undefined, f1 undefined
> (sch 'tick)
a1 #t, b1 #f, c1 #t, d1 #t, e1 undefined, f1 undefined
> (sch 'tick)
a1 #t, b1 #f, c1 #t, d1 #t, e1 undefined, f1 undefined
> (sch 'tick)
a1 #t, b1 #f, c1 #t, d1 #t, e1 #f, f1 undefined
> (sch 'tick)
a1 #t, b1 #f, c1 #t, d1 #t, e1 #f, f1 undefined
> (sch 'tick)
a1 #t, b1 #f, c1 #t, d1 #t, e1 #f,  f1 #f
>
```

All changes occur at the tick of a clock, when you call the function `(sch 'tick)`. At this point, the current inputs of a gate combine to produce the next output of the gate. Note that this is a clocked circuit and a change in the value of a connector should get propagated across only one gate at each clock. Additionally, any gate could have a delay. In this example, we assume that all and gates have a delay of 2 units.

Now let us design the system. A gate is just a function that uses the values at its input connectors to set the value of its output connector after a delay. Complete the delay unit and the and-gate code.

```
(define (delay i o)
  (define (delay-propagate)...)
  ...
  delay-propagate)


 (define (and-gate i1 i2 result)
  (define (and-propagate)...)
  ...
  and-propagate)
```

A connector maintains two values, an old value and a new value. You have to figure out why two values are required. It responds to the requests `'get-value` `'set-value` and `'set-old-to-new`.

```
 (define (make-connector)
  (define old-value 'undefined)
  (define new-value 'undefined)
  ...
  (define (me message)
    (cond ((eq? message 'get-value) ...)
          ((eq? message 'set-value!) ...)
          ((eq? message 'set-old-to-new!) ....))
          (else (error "Unknown Message - Connector" message))))
  me)
```

Now write the code for the scheduler. The scheduler responds to the message `'tick` by taking certain actions for each of the connectors and each of the gates connected to it.

```
  (define (scheduler list-of-connectors list-of-elements)
     (define (process-tick) ...)
     (define (me request)
        (cond ((eq? request 'tick) (process-tick))
           (else (error "Unknown request - Scheduler" request))))
     me)
```

Do not leave any function undefined in your code. (25 Marks)

4. All of you probably know the crypt-arithmetic puzzle:

```
      S E N D
    + M O R E
    ----------
     M O N E Y
```

Here each of the alphabets above stand for a distinct digit such that the resulting addition is correct.

Write a prolog rule: `solution(S,E,N,D,M,O,R,Y)` which will solve the puzzle. (10 Marks)

```
| ?- solution(S,E,N,D,M,O,R,Y).

D = 7 E = 5 M = 1 N = 6 O = 0 R = 8 S = 9 Y = 2 ?

(8001 ms) yes
```