Programming Paradigms
Written Exam (6 CPs)

06.07.2015

First name		Last name	
Student number		Signature	

Instructions for Students

- Write your name and student number on the exam sheet and on every solution sheet you hand in and also sign them.
- This is a closed book exam: the only resources allowed are blank paper and pens (do not use pencils).
- Write neatly and clearly. The clarity of your explanations will affect your grade.
- The duration of the exam is 2 hours.

Good luck!

Do not write in this space

Question	Marks	Achieved
1	25	
2	12	
3	8	
4	10	
5	15	
6	10	
7	8	
8	12	
Total	100	

Exercise 1 (25 marks)

a. (3 marks) Briefly describe the main difference between a compiled language and an interpreted language.

b. (3 marks) Is the following Ruby expression syntactically correct?

```
['2', 'plus', '3', 'is', "#{2+3}"].each { |x| puts x }
```

If no, explain what is wrong. If yes, what does the expression do?

c. (3 marks) Briefly describe the concept of tail recursion, and why it is desirable to write tail recursive functions.

d. (4 marks) The box model of Prolog execution is a simple way to show the control flow. Briefly sketch and describe the box model.

e. (4 marks) What is the following Prolog program doing?

```
foo :-
    repeat,
    read(X),
    write(X),
    nl,
    X = 'quit',
    !.
```

f. (4 marks) What is wrong with the following case statement in Erlang? How could you fix the code?

```
case X of
   {_,_} -> doA;
   {_,3} -> doB;
   {2,_} -> doC;
   {2,3} -> doD
end.
```

g. (4 marks) Briefly describe the following Haskell expression and show the result?

```
foldl (\x y -> y + x ) 10 [1,2,3]
```

Exercise 2 (12 marks) Assume temperature data stored in an array `t`. Write a Ruby function `maxperiod(t,x)` that calculates the length of the longest (hot) period with temperature values greater than `x`. The result is printed to the console. For example, for `t = [20, 25, 26, 23, 27]` and `x = 24`, the function prints `The longest period greater than 24 is of length 2`.

Exercise 3 (8 marks) Write a Ruby function `palindrome(a)` to check whether an array of characters represents a palindrome, i.e., is identical to the reversed array. For instance, the array `['A', 'B', 'B', 'A']` represents a palindrom, while `['A', 'B', 'B', 'A', 'C']` does not. You are not allowed to use Ruby's `array` method `reverse`.

Exercise 4 (10 marks) The product of two natural numbers can be expressed as a repeated addition using the following recursive definition (Peano axioms):

$$0 * y = 0$$
$$x * y = (x - 1) * y + y$$

Write a Prolog program to compute a product using this definition.

Exercise 5 (15 marks) Write a Prolog program `duplicates(L,D)`, which takes a list `L` of integer values and returns in `D` a list of all elements in `L` that occur at least twice; `D` should contain each duplicate value only once. For example, `duplicates([1, 2, 5, 5, 2, 4, 2])` should return `D = [2, 5]` (or `D = [5, 2]`, the order does not matter) and not `D = [2, 5, 5, 2, 2]`). You should use an accumulator to collect the elements in the result list. (Hint: if you have difficulties with the accumulator, try to write a program without an accumulator, for which you can get up to 10 marks)

Exercise 6 (10 marks) Write a function `loop` for an Erlang process that receives messages consisting of a single parameter and does the following:

- if the parameter is a number, it outputs to the console whether the number is positive, negative, or zero;
- if the parameter is "bye", the process terminates;
- otherwise, a error message is printed, e.g., "Unexpected message".

Show also how to start the process. (Hint: you can use a function `is_number(N)`, which is true if `N` is a number, and false otherwise)

Exercise 7 (8 marks) Write a Haskell function `noOfElem` that counts the number of elements in a list. Your function should return the same result as the function `length`. Do not use the function `length` for implementing `noOfElem`. You may use other functions, though.

Exercise 8 (12 marks) Write a Haskell function `diffrev`, which takes as input two lists, A and B, each containing n numbers, and produces in output a list C such that

$$\begin{aligned} C[0] &= A[0] - B[n-1] \\ C[1] &= A[1] - B[n-2] \\ &\dots \\ C[n-1] &= A[n-1] - B[0] \end{aligned}$$

For example, `diffrev [1,2,10] [2,5,-2]` returns `[3,-3,8]`. (Hint: think about breaking the problem down in more than one function!)

Solution 1

a. Compiled languages are translated into a from that can be run directly on a computer's processor. Usually the whole program is translated before it is run.

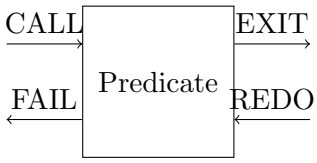
   Interpreted languages are processed by a higher-level virtual machine. Usually a program is translated on the fly, i.e., a statement is translated and then immediately executed.

b. The expression is syntactically correct. The result is: `2 plus 3 is 5`

c. A function is tail-recursive if there is no operation after the recursive call, that is, no operations are executed after the recursive call terminates. As a consequence, no data need to be stored on the stack that is needed when the recursive call terminates. Hence, different from non-tail-recursive function, for tail-recursive functions the stack does not grow with each recursive call.

d. The box model provides a simple way to show the control flow of a Prolog program. A box represents the invocation of a single predicate. The box has four ports (with associated events):

   - CALL: The first call of a predicate; control enters into the box
   - EXIT: The goal has been proven
   - REDO: The system comes back to a goal, trying ot re-satisfy it, i.e., backtracking
   - FAIL: The goal/predicate fails



e. This is a simple echo program. It reads from the standard input and shows the input on the standard output. When the user input is "quit", the program terminates.

f. The cases are in the wrong order, i.e., the most general ones come first. Consequently, the later cases will never be reached. The code can be fixed by reversing the order of the cases.

g. The function `foldl` applies a function with two input parameters to all elements of a list. The applied function has as input a list element and another parameter. In this example, the elements of the list are added to the value `10`. Hence, the return value of this expression is `16`.

Solution 2

```
def maxperiod( t, x )
  lmax = 0
  i = 0
  while i < t.length
    if t[i] > x
      l = 1
      l += 1 while i + l < t.length && t[i+l] > x
      lmax = l if l > lmax
      i = i + l
    else
      i += 1
    end
  end
  puts "Longest period with more than #{x} degrees is of length #{lmax}"
end
```

An alternative solution is

```
def maxperiod2( t, x )
  lmax = 0
  l = 0
  t.each{ |v|
    if v > x
      l += 1
    else
      lmax = [l,lmax].max
      l = 0
    end
  }
  puts "Longest period with more than #{x} degrees is of length #{lmax}"
end
```

Solution 3

```
def palindrome(a)
  if a.length == 1 || a.length == 0
    true
  else
    if a[0] == a[-1]
      palindrome(a[1..-2])
    else
      false
    end
  end
end
```

Solution 4

```
prod( 0, _, 0 ).
prod( N, M, P ) :-
    N > 0,
    N1 is N - 1,
    prod( N1, M, K),
    P is K + M.
```

Solution 5

```
duplicates( L, D ) :-
    duplicates_acc( L, [], D ).

duplicates_acc( [], A, A ).
duplicates_acc( [H|T], A1, A ) :-
    member(H, T),
    \+( member( H, A1 ) ), !,
    duplicates_acc( T, [H|A1], A ).
duplicates_acc( [_|T], A1, A ) :-
    duplicates_acc( T, A1, A ).
```

A solution without accumulator, and the result list may contain more than one occurence of each duplicate.

```
duplicates2( [], [] ).
duplicates2( [H|T], [H|D] ) :-
    member( H, T ), !,
    duplicates2( T, D ).
duplicates2( [_|T], D ) :-
    duplicates2( T, D ).
```

Solution 6

```
-module(sign).
-export([loop/0]).

loop() ->
  receive
    N when is_number(N) ->
      if
        N < 0 -> io:format("Number is negative~n");
        N > 0 -> io:format("Number is positive~n");
        N == 0 -> io:format("Number is zero~n")
       end,
      loop();
    "bye" ->
      io:format("Bye~n");
    _ ->
      io:format("Unexpected message~n"),
      loop()
  end.

P = spawn( fun sign:loop/0 ).
```

Solution 7

```
module NoOfElem where

noOfElem :: [a] -> Int
noOfElem [] = 0
noOfElem (h:t) = 1 + noOfElem t
```

An alternative solution that is tail-recursive:

```
module NoOfElem where

noOfElem2 :: [a] -> Int -> Int
noOfElem2 [] x = x
noOfElem2 (h:t) x = noOfElem2 t (x+1)
```

Solution 8

```
module Diff (diffrev) where

rev :: [Integer] -> [Integer]
rev [] = []
rev (h:t) = rev(t)++[h]

diff :: [Integer] -> [Integer] -> [Integer]
diff [] [] = []
diff (h1:t1) (h2:t2) = [h1-h2]++(diff t1 t2)

diffrev :: [Integer] -> [Integer] -> [Integer]
diffrev a b = diff a (rev b)
```