Programming Paradigms
Written Exam (4 CPs)

14.09.2015

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	10	
3	10	
4	15	
5	10	
6	10	
7	12	
8	8	
Total	100	

Exercise 1 (25 marks)

a. (5 marks) Briefly describe the main differences, advantages and disadvantages of static typing and dynamic typing.

b. (5 marks) Given is the expression `[20, "20.0", 20.0]`.

   - Is the expression legal in Ruby?
   - Is the expression legal in Haskell?

   Explain your answers and briefly describe the meaning of this expression.

c. (5 marks) Briefly describe the difference between the following expressions in Prolog:

   - `?- 7 = 3 + 4`
   - `?- 7 is 3 + 4`

   What will be the result of executing these expressions?

d. (5 marks) When you move the execution of an Erlang program from a single-processor machine to a multi-core machine or a cluster of computers, do you have to rewrite or adapt your program? Explain your answer.

e. (5 marks) Briefly explain the concept of list comprehension in Haskell and give a short example.

Exercise 2 (10 marks) Write a Ruby function `reverse(a, b)` that returns `true` if the integer array `a` is the reverse of the integer array `b`, and `false` otherwise. For example, `reverse([1,2,3,4],[4,3,2,1])` returns `true`, while `reverse([1,2,3],[4,3,2,1])` gives `false`. You are not allowed to use Ruby's `array` method `reverse`.

Exercise 3 (10 marks) The following Ruby function computes $\lceil \log_2 n \rceil$ recursively:

```
def log( n )
  return 0 if n == 1
  if n.even?
    n = n / 2
    return log( n ) + 1
  else
    n += 1
    return log( n )
  end
end
```

Rewrite this function into a tail-recursive function.

Exercise 4 (15 marks) Write a program in Prolog that goes through a list of numbers and selects numbers (starting from the beginning of the list) whose sum is smaller than a given capacity. So as long as there is still enough capacity left, the program keeps selecting numbers (skipping numbers that are too large). The program should return the result in a list. For example, given the list `[2,5,3,8,1,12]` and the capacity 14, the program should return the list `[2,5,3,1]` as this sums up to 11, which is less than 14 (when 8 and 12 are reached, they are too large to be included). The order of the items in the result does not matter.

Exercise 5 (10 marks) The following knowledge base of a Prolog program describes an electrical supply grid, organized in a hierarchical structure (i.e., there are no cycles). `p` is the power plant, `ti` stands for transformer `i`, and `homej` for home `j`. The `supplies`-clause represents a direct connection between two points.

```
supplies(p,t1).
supplies(p,t2).
supplies(t1,t3).
supplies(t1,t4).
supplies(t2,home1).
supplies(t2,home2).
supplies(t3,home3).
supplies(t4,home4).
supplies(t4,home5).
```

Write a predicate `hookedup(Y)` that tells whether `Y` is (directly or indirectly) connected to the power plant `p`, e.g., `hookedup(t3)` succeeds whereas `hookedup(t5)` fails.

Exercise 6 (10 marks) Write a function `loop` for an Erlang process that does the following. When it receives the message `"event"`, it sends a message containing the current count of events to a process registered with an atom called `observer`, displays the count on the screen, and increments the current event count. If it receives the message `"bye"`, it displays "Bye" on the screen and quits.

Exercise 7 (12 marks) Write a function `isbalanced` in Haskell that checks whether a string containing open and closed parentheses is balanced. A string of parentheses is balanced when every open one has a corresponding closed one and at any point there are not more closed ones than open ones. An empty string is balanced. For example,

- `"(())"` is balanced
- `"(())()"` is balanced
- `"()(()"` is not balanced
- `"())"` is not balanced

(Hint: a string in Haskell is simply a list of characters.)

Exercise 8 (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.

Solution 1

a.
- Static typing:
  - types and their constraints are checked before executing the program
  - pro: less error-prone
  - con: sometimes too restrictive
- Dynamic typing:
  - type checking is done during program execution
  - pro: more flexible
  - con: harder to debug

b. Ruby: This is an array containing an integer, a string, and a float. Yes, it is legal in Ruby to mix different types in the same array.
Haskell: This is a list containing an integer, a string, and a float. No, in Haskell it is not allowed to store different types in the same list.

c. In the expression `7 = 3 + 4`, the unification operator `=` tries to match the left-hand and right-hand side. This is not possible here, since `3+4` is not evaluated, hence the two sides are different, and the goal fails.
In the expression `7 is 3 + 4`, the `is` operator first evaluates the right-hand side to the value `7` and then matches it to the left-hand side; the goal succeeds.

d. No, Erlang programs don't have to be modified to run on a multi-core. The main reason is that processes do not share any resources and communicate exclusively by message passing. Therefore, it makes no difference whether they run on the same machine or on different machines. The Erlang virtual machine will automatically adapt and use the underlying hardware.

e. List comprehension in Haskell is a compact way to specify complex and possibly infinite lists by specifying an output function, a variable, an input set and a predicate.
Example: The list of even numbers between 1 and 100 can be defined as
`[x | x <- [1..100], mod x 2 == 0]`

Solution 2

```
def reverse( a, b )
  if  a.length == 0 && b.length == 0
    return true
  else
    if a[0] == b[-1]
      return reverse( a[1..-1], b[0..-2] )
    else
      return false
    end
  end
end
```

Alternative solution:

```
def reverse2( a, b )
  return true if a.length == 0 && b.length == 0 ||
                 a[0] == b[-1] && reverse( a[1..-1], b[0..-2] )
  return false
end
```

Solution 3

```
def log_tail_recursive( n )
  return log_tr( n, 0 )
end

def log_tr( n, r )
  return r if n == 1
  if n.even?
    n = n / 2
    return log_tr( n, r+1 )
  else
    n += 1
    return log_tr( n, r )
  end
end
```

Solution 4

- Solution with accumulator

```
fit( L, C, R ) :- fit_acc( L, C, [], R ).

fit_acc( [], C, L, L ).
fit_acc( [H|T], C, L, R ) :-
    H > C,
    fit( T, C, L, R ).
fit_acc( [H|T], C, L, R ) :-
    H <= C,
    N is C - H,
    fit_acc( T, N, [H|L], R ).
```

- Solution without accumulator

```
fit2( [], _, [] ).
fit2( [H|T], C, R ) :-
    H > C,
    fit2( T, C, R ).
fit2( [H|T], C, [H|R] ) :-
    H =< C,
    C1 is C - H,
    fit2( T, C1, R ).
```

Solution 5

```
hookedup(Y) :- supplies(p,Y).
hookedup(Y) :- supplies(X,Y), hookedup(X).
```

Solution 6

```
-module(eventcount).
-export([loop/1]).

loop(Counter) ->
  receive
    "event" ->
      observer ! Counter,
      io:format("event ~p~n",[Counter]),
      loop(Counter+1);
    "bye" ->
      io:format("bye~n")
  end.
```

Solution 7

```
module IsBalanced where

isbalanced :: [Char] -> Bool
isbalanced s = isbalan s 0

isbalan :: [Char] -> Int -> Bool
isbalan [] 0 = True
isbalan [] nonzero = False
isbalan (h:t) x =
   if x < 0 then
      False
   else
      if h == '(' then
         isbalan t (x+1)
      else
         isbalan t (x-1)
```

Solution 8

```
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)
```