Programming Paradigms
Written Exam (4 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	12	
7	8	
8	10	
Total	100	

Exercise 1 (25 marks)

a. (5 marks) Briefly describe the main differences between static typing and dynamic typing.

b. (5 marks) Is the following Ruby expression syntactically correct?

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

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

c. (5 marks) Briefly describe the generate and test pattern in Prolog.

d. (5 marks) What is wrong with the following case statement in Erlang? How could you fix the code?

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

e. (5 marks) Briefly explain list comprehension in Haskell and give an example.

Exercise 2 (12 marks) Write a Ruby function that takes as input an array of temperature data and computes the minimum, the maximum and the average temperature. The three values are output to the console. For example, for the input data `[20, 21, 22, 20, 17]`, the output is `min = 17`, `max = 23`, `avg = 20`. You are not allowed to use the built-in functions `min` and `max`, e.g., `[1,2,3].min` and `[1,2,3].max`.

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 program in Prolog `fit` that iterates 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 6 (12 marks) Write a function `loop` for an Erlang process that receives meassages 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 (10 marks) Write a function in Haskell that finds an element in a list. It returns `True` if the element is in the list, otherwise it returns `False`.

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.

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

d. The "generate and test" pattern has the following form:
`foo :- g1, g2, ..., gn, t1, t2, ..., tm.`

- The predicates `g1, g2, ..., gn` generate lots of different potential solutions.
- The predicates `t1, t2, ..., tm` test, whether something generated by `g1, g2, ..., gn` is a solution; if this is not the case, backtracking starts and `g1, g2, ..., gn` generate the next candidate.

e. The cases are in the wrong order, i.e., the most general ones comes in the second place, hence case three and four will never match. The code can be fixed by moving the second case to the end.

f. 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 timeseries( a )
  min = max = a[0]
  sum = i = 0
  while i < a.length
    max = a[i] if a[i] > max
    min = a[i] if a[i] < min
    sum += a[i]
    i += 1
  end
  puts "min = #{min}"
  puts "max = #{max}"
  puts "avg = #{sum/a.length}"
end
```

An alternative solution would be:

```
def timeseries( a )
  min = max = a[0]
  sum = i = 0
  a.each{ |v| max = v if v > max; min = v if v < min; sum += v }
  puts "min = #{min}"
  puts "max = #{max}"
  puts "avg = #{sum/a.length}"
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

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

A solution with an accumulator:

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

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

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 Findelement where

findelement :: Eq a => a -> [a] -> Bool
findelement x [] = False
findelement x (h:t) =
   if x == h then
       True
   else
       findelement x t
```