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

<table>
<thead>
<tr>
<th>Exercise</th>
<th>Marks</th>
<th>Achieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Exercise 1 (20 marks)

a. (4 marks) Briefly describe the concept of mixins in Ruby.

b. (4 marks) Briefly describe a key technique/construct that often helps to transform a non-tail-recursive program into a tail-recursive program (we introduced it with Prolog, but can be applied in other languages as well).

c. (4 marks) Explain the following Haskell expression and show its result?
   \((x y \rightarrow x + y)\) 10 20

d. (4 marks) Consider the Haskell function \(\text{prod} \ x \ y = x * y\). How is the function call \(\text{prod} \ 2 \ 4\) evaluated? What is the name of this evaluation concept?

e. (4 marks) What is the following Erlang expression doing and what is the result?
   \(\text{lists:map(fun(X) \to X \times X end, [1,2,3,4,5]}\)

Exercise 2 (10 marks) Assume time series data stored in an array \(a\). Write a Ruby function \(\text{maxtrend}(a)\) that returns the length of the longest positive trend (i.e., number of strictly increasing consecutive values). For example, for \(a = [3, 1, 4, 5, 7, 7, 8]\) the return value is 3, and for \(a = [3, 2, 1]\) we get 0.

Exercise 3 (8 marks) Write a class \(\text{Vehicle}\) with a field \(\text{year}\) and a method \(\text{printStats}\) that prints the value of \(\text{year}\). The value of \(\text{year}\) is given as an argument to the constructor of \(\text{Vehicle}\). Next, write a class \(\text{Car}\) that extends \(\text{Vehicle}\) and has two fields \(\text{model}\) and \(\text{brand}\). The constructor of car has three arguments, and it assigns values to \(\text{model}\) and \(\text{brand}\) and calls the constructor of the superclass \(\text{Vehicle}\) to initialize \(\text{year}\). Furthermore, create a method \(\text{printStats}\) which prints \(\text{model}\) and \(\text{brand}\), and calls the parent method in \(\text{Vehicle}\) to print \(\text{year}\). Finally, assume that there exists a module \(\text{FourWheeled}\). Use this module to create a mixin with the class \(\text{Car}\).

Exercise 4 (12 marks) Given is a directed graph \(G\), which is represented by a set of arc facts of the form \(\text{arc}(X, Y)\). Here is an example:

\[
\begin{align*}
\text{arc}(a, b).
\text{arc}(b, a).
\text{arc}(b, c).
\text{arc}(d, b).
\end{align*}
\]

Write a Prolog predicate \(\text{path}(X, Y, P)\) that computes a path \(P\) from a node \(X\) to a node \(Y\), if such a path exists. An edge can occur only once in a path, i.e., cycles should be avoided. For instance, \(\text{path}(a, c, P)\) should return \(P = [(a, b), (b, c)]\); path \(P = [(a, b), (b, a), (a, b), (b, c)]\) is not valid since it contains a cycle. Notice that the order of the arcs in the returned path does not matter, i.e., \(P = [(b, c), (a, b)]\) is also ok. (Hint: use an accumulator to collect the arcs of the path.)
Exercise 5 (10 marks) The product of two natural numbers can be expressed as a repeated addition using the following recursive definition (Peano axioms):

\[
\begin{align*}
0 \ast y &= 0 \\
x \ast y &= (x - 1) \ast y + y
\end{align*}
\]

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

Exercise 6 (12 marks) Write a Haskell function that takes as input a list of elements and a number \( k \), and returns the list, where every \( k \)-th element is removed. For instance, `dropEvery [1,2,3,4,5,6] 3` returns the list \([1,2,4,5]\).

Exercise 7 (14 marks) Write a Haskell module that exports a (tail-recursive) function `countElem` that takes as input an element \( x \) and a list and returns the number of occurrences of \( x \) in the list. For instance, `countElem 1 [1,2,3,1]` returns 2. Notice that the maximum of 14 points can only be obtained with a correct tail-recursive solution.

Exercise 8 (14 marks) Write an Erlang module with a function `stack/0` for an Erlang process that implements a simple stack data structure and offers the following functionalities upon the receipt of a corresponding message:

- “push” and an element: push element to the stack;
- “pop”: pop the top element from the stack, provided that the stack is not empty;
- “size”: determine the size of the stack;
- “reset”: reset to the empty stack.

In all cases, print a corresponding message to the console, such as the element pushed to the stack, the element popped from the stack, the size of the stack, and a reset message, respectively. Show also how to start the process.
Solution 1

a. A mixin in Ruby is a combination of modules and classes. More specifically, a module can be included in a class definition. By doing so, all methods of the module are added and available to the class. Mixins have some similarity to the concept of multiple inheritance.

b. The key idea is to compute partial/intermediate results and store them in a so-called accumulator, which is passed as an additional parameter in each recursive call. When a base case is reached, the accumulator contains the final result.

c. This is an anonymous function with two input parameters. It computes the sum of the two parameters.

d. The function is evaluated in two steps:
   - The first input parameter is applied, i.e., prod 2, yielding a partially evaluated function (\( y \mapsto 2 \times y \))
   - The partially evaluated function is applied to the second argument, i.e., (\( y \mapsto 2 \times y \)) 4, yielding 8

   This is called curried functions.

e. Calls the built-in function map, which is defined in the module lists and accepts two parameters: a function and a list. The function is applied on each element of the list. In this case, the square of each list element is computed and returned.

Solution 2

```ruby
def maxtrend( a )
x = a[0]
lmax = l = 0
a.each{ |v|
  if v > x
    l += 1
    x = v
  else
    lmax = [l,lmax].max
    l = 0
    x = v
  end
}
lmax = [l,lmax].max
return lmax
end
```
Solution 3

class Vehicle
  def initialize(year)
    @year = year
  end

  def printStats
    print "Year constructed: #@year\n"
  end
end

class Car < Vehicle
  include FourWheeled
  def initialize(brand,model,year)
    super(year)
    @brand = brand
    @model = model
  end

  def printStats
    print "Brand: #@brand\n"
    print "Model: #@model\n"
    super
  end
end
Solution 4

\[
\text{path}(X, Y, P) :- \text{path0}(X, Y, [], P).
\]

\[
\text{path0}(X, X, P, P).
\]

\[
\text{path0}(X, Y, P0, P) :-
\begin{align*}
\text{arc}(X, Z), \\
\text{not( member}((X,Z), P0)), \\
\text{append}(P0, [(X,Z)], P1), \\
\text{path0}(Z, Y, P1, P).
\end{align*}
\]

Solution 5

\[
\text{prod}( 0, _, 0 ).
\]

\[
\text{prod}( N, M, P ) :-
\begin{align*}
N &> 0, \\
N1 &\text{ is } N - 1, \\
\text{prod}( N1, M, K), \\
P &\text{ is } K + M.
\end{align*}
\]
Solution 6

module Dropevery (
    dropevery
) where

dropevery :: [a] -> Integer -> [a]
dropevery list count = dropevery2 list count count

dropevery2 :: [a] -> Integer -> Integer -> [a]
dropevery2 [] _ _ = []
dropevery2 (x:xs) count 1 = dropevery2 xs count count
dropevery2 (x:xs) count n = x : (dropevery2 xs count (n - 1))

Solution 7

module NoOfElem (noOfElem) where

noOfEl :: Eq a => a -> [a] -> Int -> Int
noOfEl x [] cnt = cnt
noOfEl x (h:t) cnt | x == h = noOfEl x t (cnt+1)
noOfEl x (h:t) cnt = noOfEl x t cnt

noOfElem :: Eq a => a -> [a] -> Int
noOfElem x lst = noOfElem x lst 0

Non tail-recursive version:

module NoOfElem (noOfElem) where

noOfElem :: Eq a => a -> [a] -> Int
noOfElem x [] = 0
noOfElem x (h:t) = (if h == x then 1 else 0) + noOfElem x t
Solution 8

-module(stack).
-export([stack/0]).

stack() -> loop([], []).

loop(S) ->
  receive
    {"push", E} ->
      io:format( "Push ~p~n", [E] ),
      loop(E ++ S);
    "pop" ->
      L = length(S),
      if
        L > 0 ->
          [H|T] = S,
          io:format( "Pop ~p~n", [H] );
        true ->
          io:format( "Stack is empty~n" ),
          T = S
      end,
      loop(T);
    "length" ->
      io:format( "Length = ~p~n", [length(S)] ),
      loop(S);
    "reset" ->
      io:format("Reset stack~n"),
      loop([])
  end.

Pid = spawn(fun stack:stack/1).