Exam Programming Languages

Date: Friday, 01.06.2018.
Duration: 90 minutes
Material: You are NOT allowed to use any material (e.g., script, exercises including solutions, notes, electronic devices...)
Number of exercises: 7
Total points: 70

Firstname, lastname: _____

Matriculation number: _____

Write your name on each extra page you deliver.
Consecutively number all pages. Total number of extra pages: _____

Exercise 1 (18 Points)

Answer the following questions. Do not write more than 3 sentences. Each question is worth 2 points.

1. Name three programming paradigms/styles and note their unique characteristics.

2. Are two following Haskell `times` functions equivalent? Justify your answer.
```
times x y = x * y
times (x,y) = x * y
```

3. What is the difference between monomorphic and polymorphic type? Is the following Haskell function monomorphic or polymorphic? Why?
```
sum [] = 0
sum (x:xs) = x + sum xs
```

4. What is a normal form of a λ expression? How does one reach it? Does the expression
   `(λ x. y)(λ f. (λ x. f (x x))(λ x. f (x x)))`
   have a normal form? Justify your answer.

5. Is it possible to define a recursive expression in λ calculus? Justify your answer.

6. What is the difference between abstract and concrete syntax?

7. What is static semantics and what is dynamic semantics?

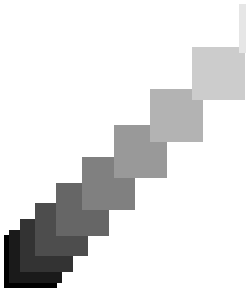
8. Explain what are parametric polymorphism and coercion and provide an example for each of them.

9. How are questions answered in a Prolog program? For example, `mother(charles, M)`.

Exercise 2 (6 Points)

A developer wants to write a program in PostScript which draws squares as in Figure 1a:

(a) Desired output



(b) Actual output

All the squares have the same dimensions, the distances between lower left corners of any two consecutive squares are the same, and each square is one nuance of a lighter grey color than the previous one, going from black to white colour. But the actual output is as in Figure 1b. Her code is as following:

```
/squares {
   /n exch def
   /size exch def
   /cy exch def
   /cx exch def
   0 1 n {
      /i exch def
      cx i 10 mul add
      cy i 10 mul add
      size
      i n div
      square
   } for
} def
/square{
   /grayScale exch def
   /size exch def
   /halfSize size 2 div def
   /cy exch def
   /cx exch def
   cx halfSize sub cy halfSize sub moveto
   cx halfSize sub cy halfSize add lineto
   cx halfSize add cy halfSize add lineto
   cx halfSize add cy halfSize sub lineto
   closepath
   grayScale setgray
   fill
} def
% usage cx cy size n squares
200 300 100 10 squares
```

Explain what is the error. Correct the code to produce the desired output.

Exercise 3 (13 Points)

NB: For this exercise, you are allowed to use only arithmetical built-in Haskell functions!

- (4 points) Write a Haskell function `magic n` which returns the magic number of the argument `n`. The magic number of a natural number `n` is equal to the product of its digits.

- (4 points) Write a Haskell function `number s` which returns the number represented by the elements of the list `s`. Consider that elements of the list are only natural numbers smaller than 10. For example, `number [1,2,3]` will return as the result the number `123`.

- (5 points) Infer the type of the following function and explain your steps. Is the function monomorphic or polymorphic? Explain.

```
apply f g [] = []
apply f g (x:xs)
   | f x < 0 = (g x) : apply f g xs
   | otherwise = (f x) : apply f g xs
```

Exercise 4 (3 Points)

Consider the following λ-expressions. Indicate which occurrences of variables are bound and which ones are free in the expressions.

- (1 point)
  `(λ x . x  y) (λ x  y  z. x) x  z`

- (2 points)
  `((λ x . λ y z . z y ) (λ x y z.  y x) y) z (λ x z. z x) (λ x . z x)`

Exercise 5 (12 Points)

We represent non-negative integers with the following Lambda expressions:

$$
\begin{aligned}
0 &\equiv \lambda f\ .\ \lambda x\ .\ x \\
1 &\equiv \lambda f\ .\ \lambda x\ .\ fx \\
2 &\equiv \lambda f\ .\ \lambda x\ .\ f(fx) \\
&\vdots \\
n &\equiv \lambda f\ .\ \lambda x\ .\ f^n x
\end{aligned}
$$

Suppose you have defined the function **if** and the operations **times**, **pred** and **isOne**. Consider the following recursive (and hence not valid) definition for the factorial calculation:

$$\textbf{fact} = \lambda n.\ \textbf{if}\ (\textbf{isOne}\ n)\ \textbf{1}\ (\textbf{times}\ n\ (\textbf{fact}\ (\textbf{pred}\ n)))$$

To do:

1. (4 points) Translate the **fact** definition into a proper definition, i.e., using the Y combinator.

2. (8 points) Write down the reduction sequence to demonstrate that factorial of 3 is 6.

Exercise 6 (8 Points)

Suppose you have a small JavaScript program with a database of animals:

```
var animal = {
   eat: function () {
       return "munch";
   }
};

var tom = Object.create(animal);
tom.name = "Tom";
tom.size = 70;

var jerry = Object.create(animal);
jerry.name = "Jerry";
jerry.size = 5;

var spike = Object.create(animal);
spike.name = "Spike";
spike.size = 100;
```

1. (2 points) What is the prototype of Tom, Jerry and Spike?

2. (3 points) Extend the code so that `tom`, `jerry` and `spike` respond to the message `sleep`. For example, `jerry.sleep()` outputs `"Jerry sleeps"`.

3. (3 points) `jerry` is sick and no longer eats, instead he responds with `"I am sick"`, while the others still respond with `"munch"`. Update the code correspondingly.

Exercise 7 (10 Points)

Create a finite collection of definite clause grammar rules to check whether a sentence is grammatically correct. A sentence can be composed of the following words: A sentence can be composed of the following words:

article `a`, `the`

noun `girl`, `girls`.

verb `play`, `plays`.

A sentence must be in the form `subject-predicate`.

- `subject` is formed out of an `article` and a `noun`. For example, `a girl`.
- `predicate` is a `verb`

The sentence should be grammatically correct in a sense that the article `a` cannot be used in front of a noun in plural. If a `subject` is in plural, the following `verb` must be `play`, and if the `subject` is in singular, the following verb should be `plays`.

Write a Prolog question to produce all correct sentences in the grammar.

You can test your program with the following examples:

```
a girl plays // True
a girl play // False
the girls plays // False
the girls play // True
girls plays // False
girls play // False
a girls play // False
```

Points

Exercise 1

Task	Points	Score
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
Total	**18**	

Exercise 2

Task	Points	Score
1	6	
Total	**6**	

Exercise 3

Task	Points	Score
1	4	
2	4	
3	5	
Total	**13**	

Exercise 4

Task	Points	Score
1	1	
2	2	
Total	**3**	

Exercise 5

Task	Points	Score
1	4	
2	8	
Total	**12**	

Exercise 6

Task	Points	Score
1	2	
2	3	
3	3	
Total	**8**	

Exercise 7

Task	Points	Score
1	10	
Total	**10**	

TOTAL

Exercise	Points	Score
1	18	
2	6	
3	13	
4	3	
5	12	
6	8	
7	10	
Total	**70**	