Programming Languages
27 June 2022

Rules

Time at disposal: 4 h.

Read carefully the following questions. For every questions, provide an answer in English. Please remember to justify why the results hold by writing the logical flow that lead you to the answers.

Note that every question can have one or more points. Points can be answered individually (no need to answer all the points of a question). To pass the exam, a perfect answer for all questions is not needed.

It is recommended to start with the questions that you believe are easy/quick, addressing the easier/quicker points and then move to the questions that you deem more elaborate. It is important to avoid prioritizing only the questions on the theory part (i.e., question 1-4) or the questions on Haskell (i.e., question 5-8). A mix of answers should be present to demonstrate a sufficient knowledge of both topics. Clearly, the more answers are provided correctly, the higher is the chance to pass and the higher will be the final grade obtained.

At the end of the exam, remember to submit the assignment to the PeerGrade system and in system DE-Digital Exam in due time. The same pdf should be submitted to both systems. Remember to avoid reporting your personal name and surname in the text submitted (the elaborate should be anonymous).

Exercises

1. Please provide an answer to the following items.

   (a) Describe what an Abstract Machine is.

   (b) Describe the functioning of an Interpreter for a generic Abstract Machine.

   (c) Describe at least two advantages of the usage of an Interpreter in comparison with a Compiler.

   (d) Describe the main disadvantage of the usage of an Interpreter in comparison with a Compiler.

2. Please provide an answer to the following items.

   (a) Describe what an Activation Record is.

(b) In the context of Memory Management, describe what the Heap is.

(c) Describe in what conditions the Stack is not enough to implement an interpreter or compiler for a language. Explain why.

(d) Describe the main advantage that the buddy system or the Fibonacci heap are offering as heap management techniques.

3. Please provide an answer to the following items.

   (a) Describe what a Type is.

   (b) When are two types equivalent if the language is using structural equivalence?

   (c) Does Haskell have structural equivalence?

   (d) Describe the goal of the unification algorithm used for performing the type inference of an expression.

4. Please provide an answer to the following items.

   (a) Consider the following pseudo code of a language using static scoping and passing parameters by name.

```
int i = 1;
int [] A = new int [5];
void fie (name int x, name int y) {
  int i = 3;
  x = x+1;
  y = 1;
  A[i] = 3;
  }
for (j = 0; j <= 4; j += 1) {
  A[j] = 0;
}
fie(i, A[i]);
write(A[1]);
write(A[2]);
write(A[3]);
write(A[i]);
```

   What does it print? Motivate your answer.

   (b) What does the previous program print if dynamic scoping is used instead of static scoping?

   (c) What does the previous program print if static scoping and call by value is used?

   (d) What happens instead if static scoping and call by reference is used?

5. Please provide an answer to the following items.

   (a) Write a function in Haskell using pattern matching that given a list of at least two integers returns 1 if the first element is 0, the second element of the list otherwise. Write also the type signature of the function.

(b) Rewrite the previous function using guards.

(c) Describe the purpose of the keyword `let` in Haskell. Define a simple expression in which the `let` keyword is used.

6. Briefly describe the data type you used to represent a configuration in your Onitama project and how you check that the player cards and the coordinates of the students pawns satisfy the lexicographic order.

7. A lot of techniques for the compression of images are based on a tree data structure called "Quad Tree". Assume that the image is square and the size of the square is a power of 2. If the image is homogeneous (same color) it is encoded, regardless of its dimension, as a leaf containing its color (see Figure 1 for a graphical representation).

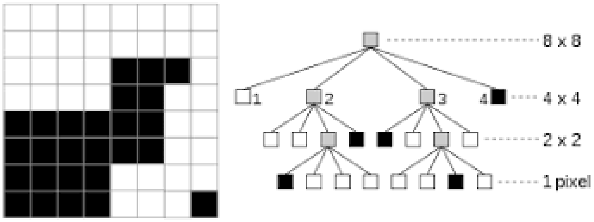

Figure 1: Graphical representation of a Quad Tree.

If the image is not homogeneous, then it is encoded as a node whose child encode i) the upper left square, ii) the upper right square, iii) the bottom right square, and iv) the bottom left square using the data type

```
data (Eq a, Show a) => QT a = C a | Q (QT a ) (QT a ) (QT a ) (QT a )
  deriving (Eq, Show)
```

Write a Haskell function `commonPoints` that given a non empty list of QuadTrees returns a QuadTrees of boolean pixels such that it has a pixel at True if in the same position all the QuadTrees of the list have the same pixels, False otherwise.

For example. Suppose that

```
z = C 0
u = C 1
q = (Q z u u u)
f = commonPoints [u, q]
```

Then `f` will output `Q (C False) (C True) (C True) (C True)`.

Write the type signature of all the functions that you define.
Motivate your choices briefly describing your code.

8. Please provide an answer to the following items.

   (a) Consider the following Haskell program

   ```
   getLine :: IO String
   putStrLn :: String -> IO ()

   main = do
     return ""
     line <- getLine
     putStrLn line
   ```

   Give the type signature of main and explain what `main` does when executed.

   (b) In the previous program, unfold the do syntactic sugar notation.

   (c) Consider the standard function mapM that can be defined on list as follows.

   ```
   mapM :: Monad m => (a -> m b) -> [a] -> m [b]
   mapM _ [] = return []
   mapM f (x:xs) =  do
     y <- f x
     ys <- mapM f xs
     return (y:ys)
   ```

   The list is an instance of Monad where

   ```
   return x = [x]
   xs >>= f = concat (map f xs)
   ```

   where concat is the function that concatenates lists that can be defined as follows.

   ```
   concat [] = []
   concat (x:xs) = x ++ (concat xs)
   ```

   Show the steps that Haskell takes to compute the following expression.

   ```
   mapM (\x -> x) [[1]]
   ```

   (d) Consider the expressions

   ```
   -- expression 1
   map (\x -> x) [[0,1],[2]]

   -- expression 2
   mapM (\x -> x) [[0,1],[2]]
   ```

   Define the type signature of the two expressions. What will we obtain by evaluating the two expressions?