Programming Languages — Homework 5
Functional programming

Due: Wednesday, 10 April 2013, 23:55

This assignment has three parts. In the first two parts, you'll explore some functional programming features of Haskell. Then, see how some functional programming ideas are used in Java.

Feel free to use G+ to ask questions, to discuss corner cases, or to post interesting test cases.

This assignment will be graded out of 10 points.

1 Functional programming in Haskell (6 pts)

1. Implement a function `find` that takes a predicate `p` (a function of type `a -> Bool`) and a list of `a` and returns a `Maybe a`—either `Nothing`, or `Just x`, where `x` is the first element of the list satisfying `p`.

   ```
   find :: (a -> Bool) -> [a] -> Maybe a
   find (>1) [1,2,3]   -- returns Just 2
   find (>3) [1,2,3]   -- returns Nothing
   ```

   Your function should not use explicit recursion. Implement the function by calling one of the list fold functions.

2. Write two functions `revl` and `revr` that behave the same as the builtin function `reverse`. `revl` should be implemented by calling `foldl`. `revr` should be implemented by calling `foldr`. Which of the two functions is more efficient?

   ```
   revl [1,2,3] == [3,2,1]
   revr [1,2,3] == [3,2,1]
   ```

Maps and folds can be defined on other recursive datatype, not just lists. A *rope* is an alterative implementation of lists (or strings) with more efficient concatenation.

```
data Rope a = List [a] | Concat (Rope a) (Rope a)
```

3. Implement `mapRope`, a map function for ropes.

   ```
   mapRope :: (a -> b) -> Rope a -> Rope b
   ```

4. Define `foldRope`, a fold function for ropes. You need to work out the parameter types of the function and then implement it.

   ```
   foldRope :: ??? -> Rope a -> b
   ```

5. Using one or both of the new rope functions above, write a function `ropeLength` that takes a `Rope a` and returns the number of a elements in the data structure. Do not use explicit recursion. matching.

   ```
   ropeLength (Concat (List [1,2,3]) (Concat (List [4,5,6]) (List []))) == 6
   ```

6. Using one or both of the new rope functions above, write a function `ropeToList` that takes a `Rope a` and returns a `[a]` with all the elements of the rope. Do not use explicit recursion.

   ```
   ropeToList (Concat (List [1,2,3]) (Concat (List [4,5,6]) (List []))) == [1,2,3,4,5,6]
   ```

2 MapReduce (2 pts)

MapReduce is a framework developed at Google for distributed computation. To use the framework, users provide two functions, a *mapper* and a *reducer*. Data consists entirely of key–value pairs. MapReduce applies the mapper function to each input pair, outputing a list of pairs. The framework then groups ("shuffles") pairs with the same key and passes sets of these pairs to the reducer function, which returns a single pair.

Google's implementation of MapReduce, of course, distributes the work on multiple machines and handles fault tolerance and load balancing issues. We're just going to simulate the algorithm in Haskell using lists and ignore the systems issues.

Write a function `mapReduce` with the following type:

```
mapReduce :: Ord c => ((a, b) -> [(c, d)]) -> ((c, [d]) -> (c, d)) -> [(a, b)] -> [(c, d)]
```

The function takes three arguments: the mapper, the reducer, and the input key–value pairs. The `Ord c` constraint is needed so that the shuffling phase can compare keys.

For example the following counts the number of word occurrences in the input list.

```
import MapReduce (mapReduce)
import Data.List.Split (splitOn)

wordCount = mapReduce makeCounts sumCounts
  where
    -- map the text to [(word, 1)]
    makeCounts (_, text) = map (\word -> (word, 1)) (splitOn " " text)

    -- reduce (word, [1,1,1,1...]) to (word, count)
    sumCounts (word, counts) = (word, sum counts)
```

The following call:

```
wordCount [
  ("gravity", "a screaming comes across the sky"),
  ("1984", "it was a bright cold day in april and " ++
           "the clocks were striking thirteen"),
  ("neuro", "the sky above the port was the color of " ++
            "television tuned to a dead channel") ]
```

should return the list of words and counts below.

```
[ ("a", 3), ("above", 1), ("across", 1), ("and", 1),
  ("april", 1), ("bright", 1), ("channel", 1), ("clocks", 1),
  ("cold", 1), ("color", 1), ("comes", 1), ("day", 1),
  ("dead", 1), ("in", 1), ("it", 1), ("of", 1),
  ("port", 1), ("screaming", 1), ("sky", 2), ("striking", 1),
  ("television", 1), ("the", 5), ("thirteen", 1), ("to", 1),
  ("tuned", 1), ("was", 2), ("were", 1) ]
```

Hint: import the `sortBy` and `groupBy` functions from `Data.List` and use these for the grouping phase of the algorithm.

Use the following file as a template:

`http://inf.usi.ch/nystrom/teaching/pl/sp13/hw/MapReduce.hs`

The following file can be used for testing.

`http://inf.usi.ch/nystrom/teaching/pl/sp13/hw/MapReduceTest.hs`

3 Functional programming in Java (2 pts)

The Java8 parallel collections library supports a functional programming style. For example, the `ParallelArray` class has a map-like method named `withMapping` and a fold-like method named `reduce`. These method take `Op` objects, which simulate first-class functions. For example, the following code maps each string to its length and then computes the sum.

```
words.withMapping(new Ops.Op<String, Integer>() {
  public Integer op(String s) {
    return s.length();
  }
}).reduce(new Ops.Reducer<Integer>() {
  public Integer op(Integer x, Integer y) {
    return x + y;
  }
}, 0);
```

The following jar files contains an implementation of the new library compiled to run on Java6 or Java7.

```
http://gee.cs.oswego.edu/dl/jsr166/dist/jsr166y.jar
http://gee.cs.oswego.edu/dl/jsr166/dist/extra166y.jar
```

Using the following file as a template:

```
http://inf.usi.ch/nystrom/teaching/pl/sp13/hw/Longest.java
```

implement a program that finds and prints the longest words in the input file. The code for identifying words is already implemented, you need only fill in the `longest` method.

To compile from the command-line with `javac` and to run with `java`:

```
javac -classpath jsr166y.jar:extra166y.jar Longest.java
java -classpath jsr166y.jar:extra166y.jar:. Longest < input
```

You are welcome to use alternative compilers and IDEs.

Documentation for the library is here:

```
http://gee.cs.oswego.edu/dl/jsr166/dist/extra166ydocs
http://gee.cs.oswego.edu/dl/jsr166/dist/extra166ydocs/extra166y/ParallelArray.html
```

You should not write

Submission

1. Complete the survey linked from the course web page after completing this assignment.
2. Submit your code and solutions on Moodle by 23:55 on Wednesday, 10 April 2013. Include your name in each file you submit.