Homework 03

Due 2/13/2019

Purpose:

The purpose of this homework assignment is to help you comprehend the concepts and properties of regular languages, especially the decision problems of regular languages. You will also write your first Haskell programs.

You will turn in this week's homework a bit differently than in the past. You will write up your complete solutions (including Haskell code) using LaTeX as usual and turn it in as Homework 3. However, I also want you to pull out your Haskell programs (answers to problem 7) and turn them in a text file named Hmwk3.hs. The first line of that file should be

```
module Hmwk3 where
```

The next line should be a comment with your name. The rest of the file should be your Haskell code. We will take that file and automatically test it by running your code on our test data. As a result, you should be sure that your code compiles properly (i.e., running "ghci Hmwk3.hs" should compile your code without errors and it should be possible to run your code on my test cases (ideally without it blowing up when executed!).

While your pdf file will be turned in to gradeScope as usual, you will need to turn in the file with programs only separately (gradescope only seems to take pdf files).

Before you submit your Haskell program file, go to `https://submit.cs.pomona.edu/2019sp/cs101` and enroll yourself in the class. When you are ready to turn in your homework, click on the assignment hw03 on that page and follow the instructions to submit your work. All of your Haskell code should be in the file Hmwk3.hs as described above.

Be sure to test your programs one last time before submitting. I've seen students mess up when commenting code in a way that causes everything to break. Code that doesnt compile will get very little credit. We will be using some automatic testing, and code that doesnt compile brings everything to a halt. With a large class we will not have time to go in to manually tweak your code to make it work.

Your text file should compile without errors in ghci. This means that in particular, problem numbers and any other non-program text should be given as Haskell comments. Single line comments start with `--`. The comment extends from wherever in the line the `--` occurs to the end of the line. You can also put in block comments. They extend from `{-` to a closing `-}`. For example,

```
silly n = n * n - 3  -- this is a silly comment on a silly function

{-  This is the start of a long
     and important comment
     telling about some code.
-}
```

IMPORTANT: When you write the functions requested below, please make sure that they have the exact names and types specified in the question. If you make an error, your program will crash on our test suite and you will get very little credit.

Be sure that all of my sample code works, and worry about edge cases that might cause your program to give the wrong answer as I will test it more thoroughly.

To summarize, this week you will turn in two separate files, each in a different way. The first is a pdf file that contains your complete solutions, and it will be turned in via gradescope. It is the one we will look at for grading and all the grades will be associated with that file. It is turned in to Homework 3. The second is the hs file containing executable code. It will be turned in on the submit.cs.pomona.edu site. It will not be returned to you – we will just download it to test your code and assign grades that will show up associated with the pdf.

Tasks:

**Problems from the texts are given in the form c.n(k) where c is the chapter and n is the problem number, and k is the sub-problem number. Thus problem 2.7(d) is problem 7(d) from Chapter 2.*

1. (0 points) **Academic Honesty Declaration** (1 page)
2. (5 points) **Regular expressions** (1 page)

   Let $\Sigma = \{a, b\}$, and let A be the set of words w over Σ for which w contains an even number of a's or an odd number of b's. Give a regular expression for the language A.
3. (10 points) **Pumping Lemma, etc.** (1 page for each sub-question)

   Problem 8.1(n, s)
4. (3 points) **Regular Languages** (1 page)

   Let $G = (V, \Sigma, R, S)$ where $V = \{a, b, S\}$, $\Sigma = \{a, b\}$, and R contains:

   - $S \to aSb$
   - $S \to aSa$
   - $S \to bSa$
   - $S \to bSb$
   - $S \to \epsilon$.

   Show $L(G)$ is regular.

   hint: The above is a context-free grammar, NOT a regular grammar. However, find a way to prove it is actually a regular language.
5. (6 points) **Minimization algorithm** (1 page for each sub-question)

   In this problem you will be creating (and proving correct) an alternate algorithm for minimizing finite state machines.

   Let $M = \{K, \Sigma, \delta, s, A\}$ and for each $q \in K$, let $M_q = \{K, \Sigma, \delta, q, A\}$ (that is, a machine that differs from M only by changing the start state). For $k \geq 0$, say states p and q of M are *k-distinguishable* if there is some input string of length at most k which is accepted by one of M_p and M_q, but not the other. Otherwise they are *k-indistinguishable.* Two states are equivalent if they are k-indistinguishable for every k.

   (a) Suppose M is a deterministic finite state machine in which all states are reachable. Show that if M has two equivalent states, one can be eliminated; but if all pairs of distinct states are inequivalent, then the machine is as small as possible.

   (b) Show that states p and q are k-indistinguishable if and only if

      i. both are accepting or both are non-accepting, and

      ii. if $k > 0$, then for each $a \in \Sigma$, $\delta(p, a)$ and $\delta(q, a)$ are $(k-1)$-indistinguishable.

   (c) Define a series of equivalence relations $\equiv_0$, $\equiv_1$, ... on the states of M as follows:

- $p \equiv_0 q$ if and only if p and q are accepting or both are non-accepting.
- $p \equiv_{k+1} q$ if and only if $p \equiv_k q$ and, for each $a \in \Sigma$, $\delta(p,a) \equiv_k \delta(q,a)$.

Show that $p \equiv_k q$ if and only if p and q are k-indistinguishable.

(d) Show that there is an $n \leq |K|$ such that $\equiv_n$ and $\equiv_{n+1}$ are identical. Then by the previous part and the definition of equivalent states, all equivalent states can be discovered by carrying through the inductive definition of $\equiv_{k+1}$ at most $|K|$ times. Use part a of this problem to describe an algorithm to minimize a given deterministic finite state machine.

6. (20 pts) **Haskell Programming** (1 page)

For this problem, use the ghci interpreter on the computers in the computer lab (or on your own computer). To run the program in the file "example.hs", type

```
-> ghci
Prelude> :l example.hs
Prelude> :set +t       -- gives more info when you type an expression
```

at the command line.

The Haskell compiler will process the program in the file and then wait for the user to type an expression. For example, if "example.hs" contains

```
-- double an integer
double x = x + x;

-- return the length of a list
listLength [] = 0
listLength (l:ls) = 1 + listLength ls
```

You can test the program by typing the following:

```
*Main> double 10
20
it :: Integer
*Main> listLength (1:[2,3,4])
4
it :: Integer
```

Start early on this part so you can see the TA or me if you have problems understanding the language. Looking at the examples in the on-line tutorials and text and in your notes will help a great deal in understanding how to use Haskell. Use pattern matching where possible.

(a) **Basic Functions**

Define a function `sumSquares` that, given a nonnegative integer n, returns the sum of the squares of the numbers from 1 to n:

```
- sumSquares 4;
30
- sumSquares 5;
55
```

Define a function `listDup` that takes a pair of an element, e, of any type, and a non-negative number, n, and returns a list with n copies of e:

```
> listDup("moo", 4);
["moo","moo","moo","moo"]
it :: [[Char
> listDup(1, 2);
[1,1]
it :: [Integer]
> listDup(listDup("cow", 2), 2)
[["cow","cow"],["cow","cow"]]
it :: [[[Char]]]
```

Your function will have a type like `(Eq t, Num t) => (a, t) -> [a]`. What does this type mean? Why is it the appropriate type for your function.

(b) **Lists**

Define a recursive function `evens` that returns the list of even numbers that occur in an input list of numbers. For example, `evens [1,2,3,4,8,5,2]` should return [2,4,8,2]. Notice that the elements are returned in the order encountered and duplicates are included. Evens has type

```
evens :: Integral t => [t] -> [t]
```

Define a function `incBy n lst` with type given by `incBy :: Num t => t -> [t] -> [t]`. This function should take parameters n and lst and then increments all of the elements of the list `lst` by n. Thus `incBy 7 [1,4,2]` should return `[8,11,9]`

For extra credit write `incBy'` that behaves exactly as above, but is defined using the built-in `map` function instead of using recursion.

Criteria:

Your assignment will be graded based on the correctness and the completeness of your solutions. Please make sure that you use the correct terminologies when you write proofs or describe a structure, such as CFG, PDA, etc.

Submission Guideline:

Please edit your HW following the editing guideline, especially the instructions on the number of pages each question should occupy. The text in blue are instructions; they should be either removed, or replaced with proper contents and turned into black. Please submit your homework solutions online via gradescope.