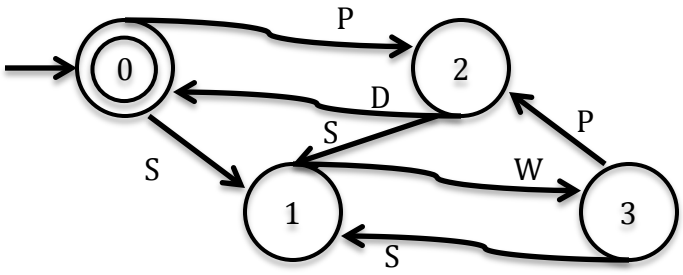
CSE116 Problem Set #4

For each of the following problems you should start by writing several JUnit tests that will verify correct behavior. Once you have written at least four distinct tests for each of the problems, begin to implement solutions to each of the problems. Be sure to run your unit tests often. They are not necessarily in order of difficulty.

For all of these problems be sure you understand how to implement a state machine – your TA will discuss the desired approach during recitation this week.

1. Define a method named accept which takes a String as argument and returns a boolean. The String consists of a sequence of characters drawn from the set { 'D', 'P', 'S', 'W' }. These characters represent "dry", "paint", "sand" and "wipe", respectively. Implement the following finite state machine:



2. Define a method named accept that accepts a String as input and returns true if the String represents a valid telephone number format, false otherwise. Valid formats are:
   a. (DDD) DDD-DDDD
   b. DDD-DDDD
   c. +1-DDD-DDD-DDDD

   Valid examples are "(716)688-0315", "645-3180" and "+1-716-645-3180". If the String does NOT represent a telephone number in a valid format, return false.

   USE A FINITE STATE MACHINE APPROACH. START BY DRAWING THE MACHINE.

3. Define a method named accept that has a single String parameter and which returns a HashMap<String,Integer>. Supposing the parameter is named inputFilePath, the method must:
   a. read the contents of a file (identified by inputFilePath) one char at a time,
   b. segment the input into words, and
   c. keep track of word counts in a java.util.HashMap<String,Integer>.

   A word is defined as a contiguous sequence of characters that does not contain word separator characters, where word separator characters are: ' ' (space), '\t' (tab), '\n' (newline), ',' (comma) and '.' (period).

   Think carefully about this definition, and how it applies to an initial character sequence not preceded by a word separator, and also a final character sequence not followed by a word separator. You must use only CharacterFromFileReader[1] to read characters from the input file. Use it as an Iterator<Character>, keeping in mind the autoboxing and unboxing features of the Java language.

   In order to keep your code readable, break your code into several methods. Define meaningful private helper methods that you call from the required public method.

[1] The definition of this class is found in the PracticeProblemSets project in the repository.