UNBC

Laboratory Assignment

Computer Science 101—Winter 2016

Searching Machines

Purpose

To practise systematizing code; preparation for interfaces.

Due Date

The completed lab assignment is due Friday, 2017-01-27 *by the beginning of lecture*.

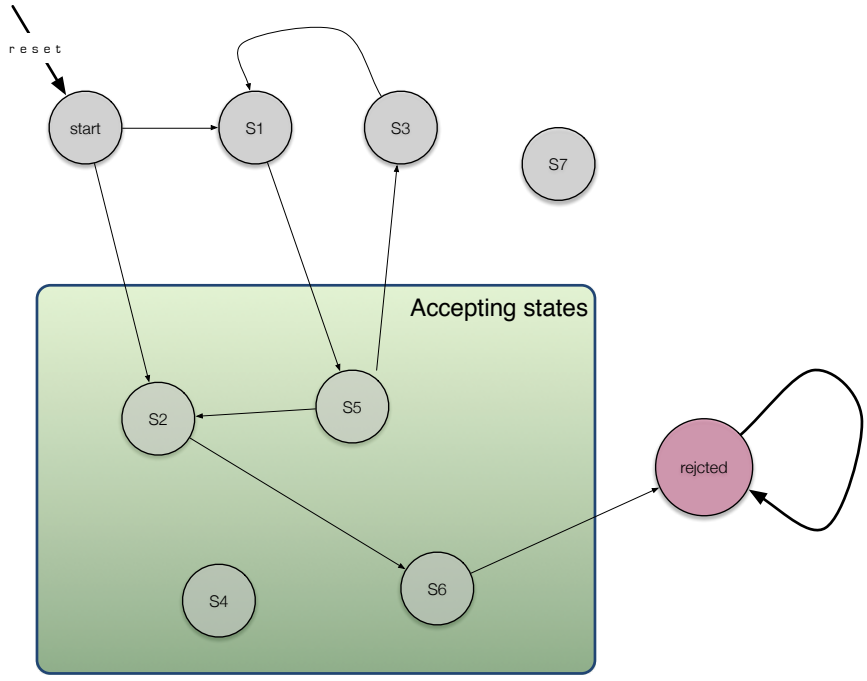

Figure 1: State Machine

Finite State Machines

Finite state machines are a mathematical abstraction studing in CPSC 242 and CPSC 340. See Figure 1.

- `void reset()` — reset the machine back to the beginning state so that we can use it again.
- `void addLetter(char c)` — add a character `c`, and change state appropriately.
- `boolean isAccepting()` — is the machine currently in an accepting state?
- `boolean isRejected()` — is the machine currently stuck in a non-accepting state? (meaning that we can reject the word we are checking).

Figure 2: 101 Machine methods

A finite state machine has a certain fixed number of states (circles in Figure 1) including a special start state. It allows has lines between states labelled by characters (the character labels are not shown in Figure 1). Starting from the start state, it accepts characters one by one. Each time it accepts a character, its state changes (or possibly stays the same) based on the transition lines.

Some of the states are *accepting states*. If the machine is in an accepting state, then it accepts the word that got it into that state. Note that it is possible to go from an accepting state to an undecided or rejecting state.

Some states are *sinks*, meaning that once you get into them you can't get out. Once you are in a non-accepting sink, you know that regardless of what characters remain, the word is going to be rejected.

Finite state machines are excellent devices for recognizing certain kinds of patterns called *regular expressions*. (See the Oracle documentation of the `String.replaceAll` and `java.util.regex.Pattern` for examples of the use of regular expressions.)

In this laboratory assignment we'll use a generalization of finite state machines.

CPSC 101 Machines

We can encapsulate the important ideas of a finite state machine in four methods, as shown in Figure 2.

The task in this lab assignment is to build machines with the methods shown in Fig-

ure 2 that recognize the words that you searched for in the previous lab assignment.

⇒ Write a stand-alone class whose objects are machines that recognize Lab 2 words (words that contain six or more consonants, all in *non*-increasing order).

Although you may not yet see the reason to do so, make sure that your class can create independent objects, and that the methods in Figure 2 are all non-static.

⇒ Re-implement Lab 2 using your machine class to recognize words.

"Degenerative" words

Some words — for instance, "comparative" and "demipriest" — contain six or more consonants, all in increasing order. (for the purposes of this lab assignment, consonants are letters that are neither vowels, nor 'y'.)

⇒ Write another stand-alone class whose objects are machines that recognize "degenerative" words. (words that contain six or more consonants, all in *strictly increasing* order).

⇒ Repeat what you did in Lab 2, but for "degenerative" words, storing the result in a file **`luminarist.txt`**.

"Abstemious" words

Some words — for instance, "abstemious" and "facetious" — contain all of the vowels exactly once, and in order. That is to say, there is one 'a', one 'e', one 'i', one 'o', and one 'u', and they occur in exactly that order.

⇒ Write yet another stand-alone class whose objects are machines that recognize "abstemious" words. Repeat what you did in above, but for "abstemious" words, storing the result in a file **`abstemious.txt`**.

One big program

⇒ Now combine your machines and logic into one program that creates the three files **`tessararian.txt`** **`luminarist.txt`**, and **`abstemious.txt`**.