CMPU 331 · Compilers · Fall 2019

Assignment 1

Due Tuesday, September 17, 2019

This assignment covers lexical analysis, regular languages, and finite-state machines. You may discuss this assignment with other students and work on the problems together. However, your write-up should be your own individual work, and you should indicate in your submission who you worked with, if applicable. Assignments should be submitted through Moodle as a PDF by 11:59pm. (You can export a PDF from Google Docs, Word, LibreOffice, LaTeX, or any other text editor you prefer.)

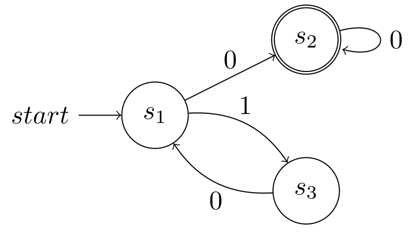
Problem 1

Write regular expressions for the following languages over the alphabet $\Sigma = \{a, b, c\}$:

1. The set of all strings that repeat the sequence abc.
2. The set of all strings that start with at least one a character, followed by any number of b characters (or none).
3. The set of all strings that contain one or more of the characters a, b, or c, combined in any order.

Problem 2

Informally describe the language accepted by the following finite-state machine, and give three examples of strings it would accept.



Problem 3

Imagine a programming language that is a simple subset of Scheme/Racket, consisting only of the tokens shown in the brief code example below.[1] Provide a complete list of tokens you would define for this language. With each token, indicate the lexeme that corresponds to the token, or describe the pattern for the token (either informally or using regular expressions). There is no one right way to name tokens, but pick names that are clear and meaningful.

[1] Code example from: ”Introduction to Computer Science via Scheme, Part 1” by Luke Hunsberger, page 139-140.

```
;; SUM-ALL
;; --------------------------------------------------
;; INPUT: LISTY, a list of numbers
;; OUTPUT: The sum of all the elements of LISTY

(define sum-all
  (lambda (listy)
    (cond
      ;; Base Case: LISTY is empty
      ((null? listy)
        ;; The sum of all the elements of the empty list
        0)
      ;; Recursive Case: LISTY is non-empty
      (else
        ;; The recursive function call computes the sum of all
        ;; the numbers in the rest of LISTY; we just add on the
        ;; first element.
        (+ (first listy) (sum-all (rest listy)))))))

(sum-all '(1 2 3 4))
(sum-all '(1 10 100 1000))
(sum-all '(2 5 3 8 1))
```