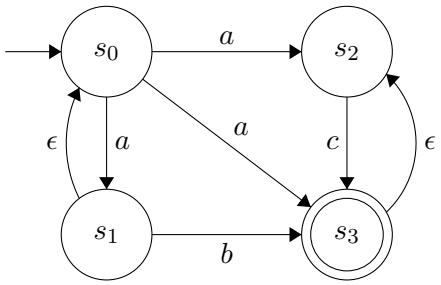
CS143 Spring 2020 - Written Assignment 1

Due Thursday, April 23, 2020 at 11:59 PM

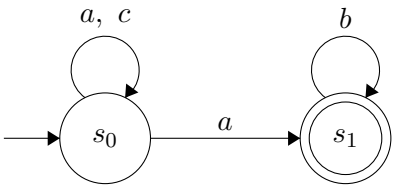
This assignment covers regular languages, finite automata and lexical analysis. You may discuss this assignment with other students and work on the problems together. However, your write-up should be your own individual work, and you should indicate in your submission who you worked with, if applicable. Assignments can be submitted electronically throught Gradescope as a PDF by 11:59 PM PDT. A LaTeX template for writing your solutions is available on the course website. There is a post on Piazza describing how to create the finite automata diagrams.

1. Write regular expressions for the following languages over the alphabet $\Sigma = \{0, 1\}$:
   (a) The set of all strings where no two consecutive characters are the same.
   (b) The set of all strings representing a binary number that is a power of 2. Allow for leading zeros e.g. 001000.
   (c) The set of all strings containing at least one of 1110 or 1011.
2. Draw the DFAs for each of the languages from Question 1.
3. Using the techniques covered in class, transform the following NFAs with ϵ-transitions over the given alphabet Σ into DFAs. Note that a DFA must have a transition defined for every state and symbol pair, whereas a NFA need not. You must take this fact into account for your transformations. Hint: Is there a subset of states the NFA transitions to when fed a symbol for which the set of current states has no explicit transition?

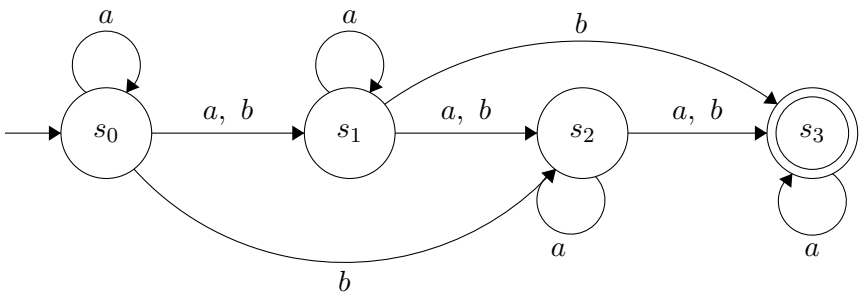
   (a) $\Sigma = \{a, b, c\}$



   (b) $\Sigma = \{a, b, c\}$



(c) $\Sigma = \{a, b\}$



4. Let L be the language over $\Sigma = a, b, c$ where the following holds: w is in L if at most one character has more than one occurrence in w.

   Examples of Strings in L: a, baaaaaac, ϵ
   Examples of Strings **not** in L: ccbb, abcab

   Draw an NFA for L.

5. Consider the following tokens and their associated regular expressions, given as a **flex** scanner specification:

```
%%
(01|10)                 printf("snake")
0(01)*1                 printf("badger")
(1010*1|0101*0)         printf("mushroom")
```

   Give an input to this scanner such that the output string is $(\texttt{badger}^{11}\texttt{mushroom}^{2})^{4}\ \texttt{snake}^{3}$, where $\texttt{A}^{\texttt{i}}$ denotes `A` repeated `i` times. (And, of course, the parentheses are not part of the output.) You may use similar shorthand notation in your answer.

6. Recall from the lecture that, when using regular expressions to scan an input, we resolve conflicts by taking the largest possible match at any point. That is, if we have the following **flex** scanner specification:

```
%%
do                          { return T_Do; }
[A-Za-z_][A-Za-z0-9_]*      { return T_Identifier; }
```

   and we see the input string "dot", we will match the second rule and emit `T_Identifier` for the whole string, not `T_Do`.

   However, it is possible to have a set of regular expressions for which we can tokenize a particular string, but for which taking the largest possible match will fail to break the input into tokens. Give an example of a set of regular expressions and an input string such that: a) the string can be broken into substrings, where each substring matches one of the regular expressions, b) our usual lexer algorithm, taking the largest match at every step, will fail to break the string in a way in which each piece matches one of the regular expressions. Explain how the string can be tokenized and why taking the largest match won't work in this case.