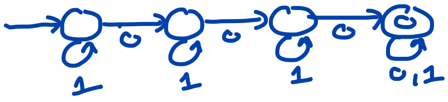
CSC 404 - Activity/Project 5 - Name:

Remark 1. Going for a victory lap on Activities 3 and 4 - i.e., let's revisit these finite automaton and write them as Regular Expressions – wee!

Problem 1. Write the language as a regular expression. For at least one, type up the example in RegExr.com (or other site) and verify the accepted and rejected list. You can take a screen shot of it working or be fancy and include a link in your dropbox submission. For example, `0*10*(10*10*)*` or `0*10*((10*){2})*` gives bit strings with an odd number of 1s. The following RegExr link shows it accepting and rejecting various input strings: https://regexr.com/5l8sr. Here we typed the RegEx with anchors ^ and $ and set the parameters to multiline. (Note - when you load the link it may kick out a *timed out* error. Go into the RegEx and retype any portion of it – this should cause it to reload.)

a. Bit strings that contain three OR MORE: 0s.

Accepted: `000, 01010, 11100110, 1111110111111011101` Rejected: `1, 00, 101011, 001111111.`

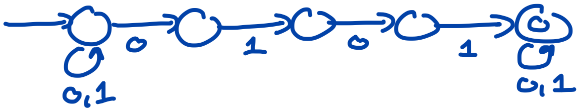


`1*01*01*0[01]*`

`(1*0){3}[01]*`

b. Bit strings that contain the substring `0101` (i.e., `w` = `x0101y` for some `x` and `y`).

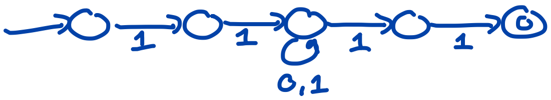
Accepted: `0101, 11010111, 000100101, 0110101` Rejected: `010, 1010, 001101, 01001.`



`[01]*0101[01]*`

c. Bit strings of at least 4 characters that begin and end with `11`.

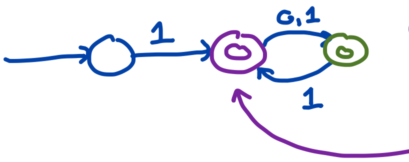
Accepted: `11111, 11011, 110011, 11101011` Rejected: `111, 1011, 000, 11101101.`



`11[01]*11`

d. Bit strings that that begin with a `1` and every other position is a `1`.

Accepted: `1,10101, 1110, 101110, 11101111111` Rejected: `10000, 110, 10110, 01010.`



$\underbrace{\texttt{1([01]1)*}}$ $\underbrace{\texttt{[01]?}}$

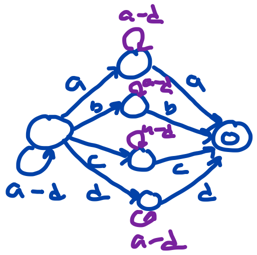
ends you here

can advance to ◎.

e. Strings over the alphabet $\{a, b, c, d\}$ such that the final character has appeared before. For example,

Accepted: `abca, bcdaa, dad, bbb` Rejected: `abc, cbbbbbba, adc, abcd.`

Note: `[a-d]` gives the set of letters a through d.



`[a-d]*(a[a-d]*a|b[a-d]*b|c[a-d]*c|d[a-d]*d)`

If we allow capture groups we can run it as

`[a-d]*([a-d])[a-d]*\1`

matches what is between ( ) '1st' set of ( )

For a-z

`[a-z]*([a-z])[a-z]*\1`

OR

`([a-z])+[a-z]*\1`