Braylee Curren

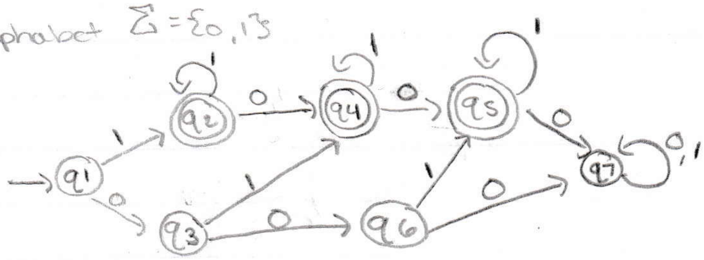
CS 397 Hw03

DFAs

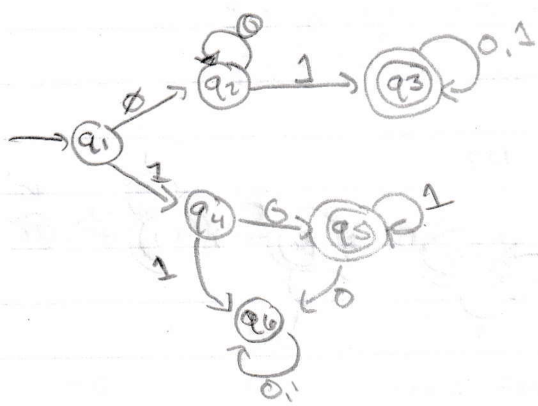
Create a DFA that recognizes the following language

{w| w contains at least one 1 + at most two 0's}

Alphabet $\Sigma = \{0,1\}$



What language does the following DFA recognize?



- starts w/ Ø
  - needs a 1 then accepted

- starts w/ 1
  - no 11 or invalid
  - 10 followed by any # 1's valid
  - 1011 0 invalid if sees another Ø

language: if it starts w/ Ø contains at least one 1
if it starts w/ a 1, 2nd must be zero and no other Ø seen

L(m): {w | w starts w/ '10' and has no other '0' or w starts w/ '0' and has at least one '1'}

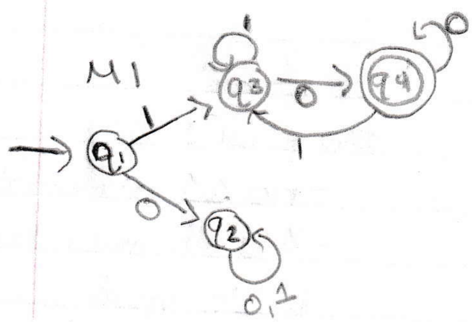
Create DFA :

Create a DFA that is the union of the following 2 languages

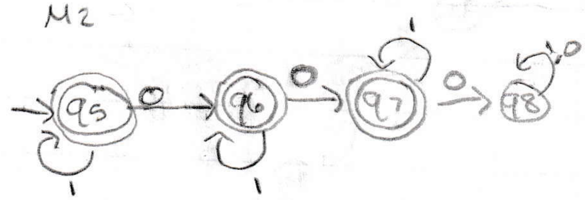
M1 $\{ w \mid w$ begins w/ 1 and ends w/ a 0$\}$

M2 $\{ w \mid w$ contains at most two 0's$\}$

M1



M2

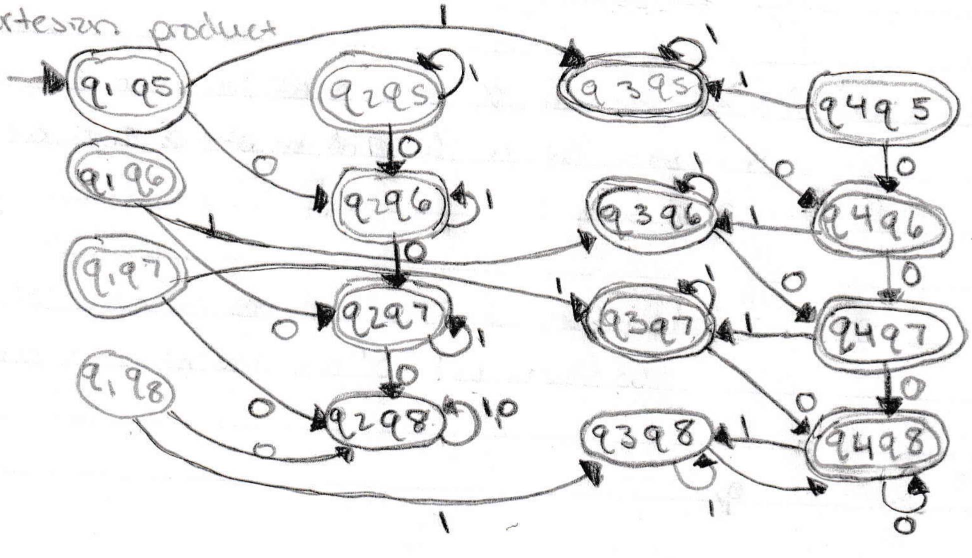


(2)- Start = node w/ both start states
q1q5

(3) Transitions = arrows

(4) Accept states where either excepted

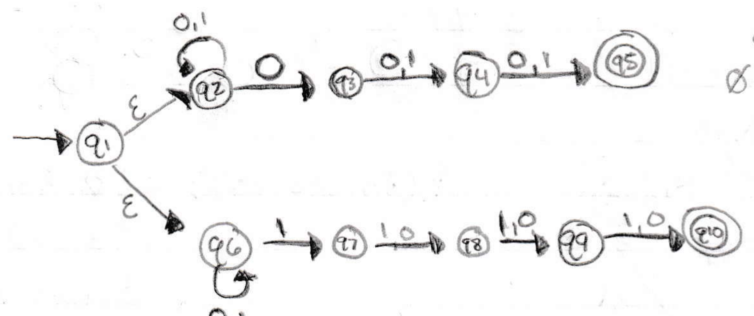
(1)
-Cartesian product



NFA

Create NFA w/ following language

{w | w has symbol 0 two spots from the end or 1 three spots from the end.

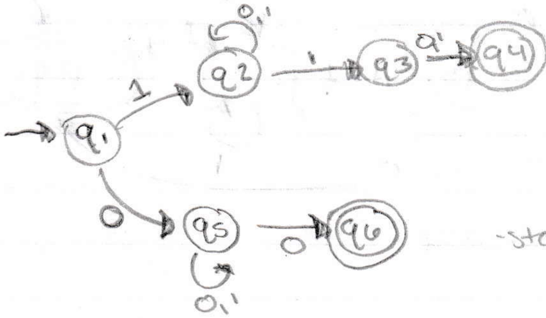


01010 01

∅ two spots from end

0101011 0 0   1 three spots from end ?

What language does the following NFA recognize?



Starts w/ 1 has 1 in spot before the last

1010101 1

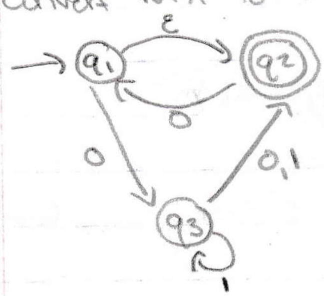
-starts w/ a zero ends w/ a zero

0 111010100

language : {w | w starts with a 1 and has a 1 the symbol before the last or w starts with a zero and ends w/ a zero}

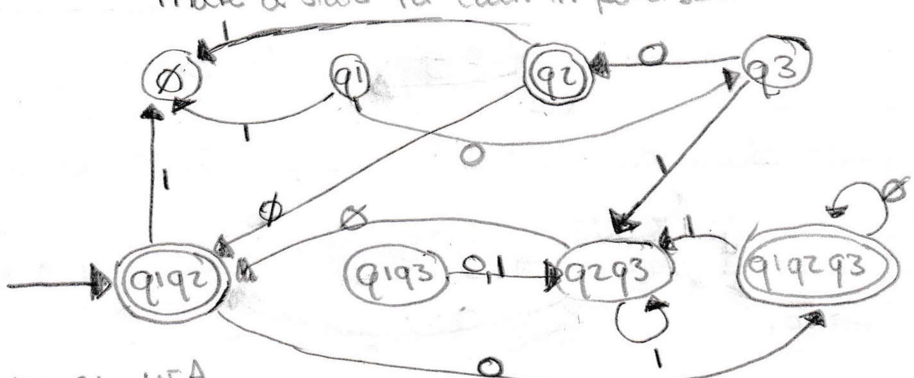
Convert to DFA

Convert NFA to DFA



$P(\{q1, q2, q3\}) = \{\emptyset, \{q1\}, \{q2\}, \{q3\}, \{q1, q2\}, \{q1, q3\}, \{q2, q3\} \{q1, q2, q3\}\}$

make a state for each in powerset



② start state same as NFA

③ accept state: Any state containing an accept state

④ add transitions

-remember ε

Regular Expressions

Create a regular expression that recognizes the following language

{w|w contains at least two 0's + at least one 1}

001   010   100 . → three combos possible

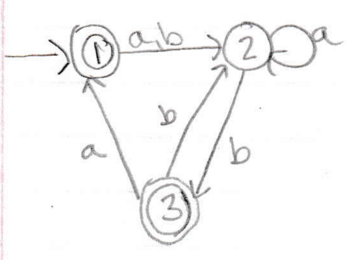
$(\Sigma^* ø \Sigma^* ø \Sigma^* 1 \Sigma^*) \cup (\Sigma^* ø \Sigma^* 1 \Sigma^* ø \Sigma^*) \cup (\Sigma^* 1 \Sigma^* ø \Sigma^* ø \Sigma^*)$

What language does the following RE recognize

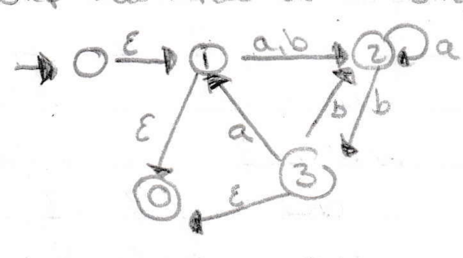
$(1(ø \cup 1)^* 1) \cup (ø(ø \cup 1)^* 1) \cup (1(ø \cup 1)^* 0)$

{w|w starts with a 1 and ends with a 1 or starts with a ø and ends in a 1 or starts with a 1 and ends in a a ø .

Convert DFA to RE

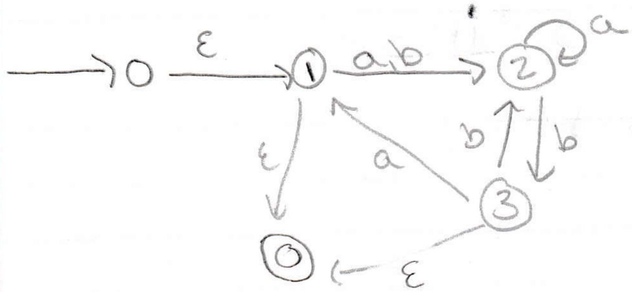


Step 1) add new start + single accept



(R1) (R2)* (R3) ∪ R4

① require start state goes to every state
② every state has transition from every state
③ every other state has transition to every other state and to itself
- leaving off ε to keep the image cleaner



qrip 2

Condition 1: $q_i = 3$ $q_j = 3$

$R1 = b$ $R2 = a$ $R3 = b$ $R4 = \varepsilon$

$(R1)(R2)^*(R3) \cup (R4)$

$b(a)^*b$ → new edge 3 to 3

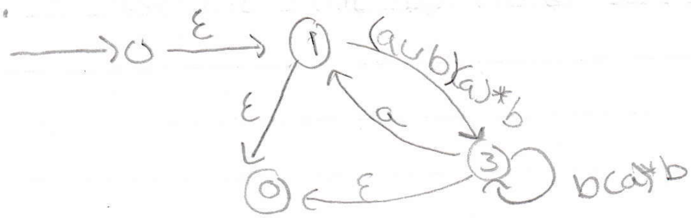
Condition 2: $q_i = 1$ $q_j = 3$

$R1 = a \cup b$ $R2 = a$ $R3 = b$ $R4 = \varepsilon$

$(R1)(R2)^*(R3) \cup R4$

$(a \cup b)(a)^*b$ new 1 to 3

Result:



qrip 3 $q_i = 1$ $q_j = 1$

$R1 = a \cup b(a)^*b$ $R2 = b(a)^*b$ $R3 = a$ $R4 = \varepsilon$

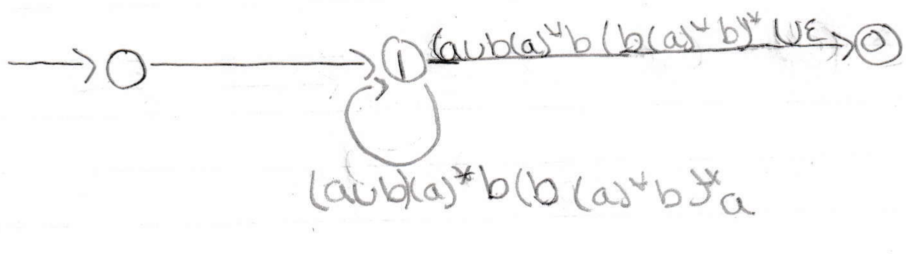
$(R1)(R2)^*(R3) \cup (R4)$

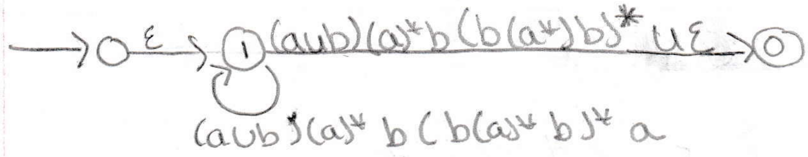
$(a \cup b(a)^*b)(b(a)^*b)^*a$ self loop 1

$q_i = 1$ $q_j = \text{end}$

$R1 = a \cup b(a)^*b$ $R2 = b(a)^*b$ $R3 = \varepsilon$ $R4 = \varepsilon$

$(a \cup b(a)^*b)(b(a)^*b)^* \cup \varepsilon$ ← important b/c it accepts empty string





grp 1

q_i = start q_j = end

$R1 = \varepsilon$ $R2 = (a \cup b)(a)^* b(b(a)^* b)^* a$

$R3 = (a \cup b)(a)^* b(b(a)^* b)^* \cup \varepsilon$ $R4 = \varepsilon$

$(R1)(R2)^*(R3) \cup R4$

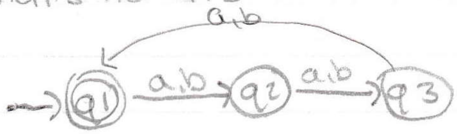
$$\underbrace{((a\cup b)(a)^* b(b(a)^* b)^* a)^*}_{\text{accept state 1}}(\underbrace{(a\cup b(a)^* b(b(a)^* b)^*}_{\text{accept state 3}} \underbrace{\cup\, \varepsilon}_{\text{accepts empty}})$$

final expression:

$((a \cup b)(a)^* b(b(a)^* b)^* a)^* ((a \cup b)(a)^* b(b(a)^* b)^* \cup \varepsilon)$

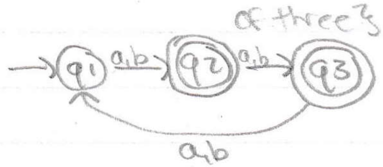
Complement:
Show that the class of regular languages is closed under complement

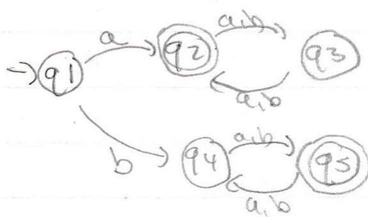
- The complement of a set S, written $\bar{S}$, is the set of all elements not in S



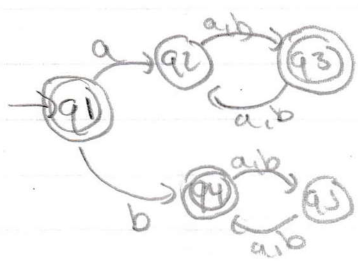
orignal {w | w has a length that is a multiple of 3}

complement {w | w does not have a length that is a multipn of three}





orignal {w | w has length @least 1 + if w/ starts w/ 'a' it has odd length + if w starts w 'b' it has even length}



complement: w can be shorter tun 1 starts w/ a even length starts w/ b odd length

The compliment of a DFA can be created by changing all accept states to non-accept states, and changing all non-accept states to accept states. This can be seen in the two examples above.
Since we have a way to change a DFA to accept its complement, regular languages are closed under compliment

Non-Regular

Problem 1.53 from page 91 in textbook

Let $\Sigma = \{0, 1, +, =\}$ and

$ADD = \{x = y+z \mid x, y, z$ are binary integers, + x is the sum of y + $z\}$.

Show ADD is not regular

Claim: $ADD = \{x = y+z \mid x, y, z$ are binary integers and x is the sum of y and $z\}$ is a regular language.

Assume for contradiction that ADD is regular

Then let p be the pumping length given by the pumping lemma

$\forall s \in ADD$ such that $|s| > p$

By the pumping lemma, s can be divided into xyz such that

$xy^iz \in ADD$ for $i \geq 0$ since ADD is regular

$S = (01011\ldots) = (00101\ldots) + (011100\ldots)$

Case 1: y has the equal sign

$$S = xyz = \underbrace{(\ldots01011\ldots)}_{x}\ \underbrace{=(\ldots01101\ldots)}_{y}\ \underbrace{+(\ldots011100\ldots)}_{z}$$

Then xy^2z must be in ADD

$$\underbrace{(\ldots01011\ldots)}_{x}\ \underbrace{=(\ldots01101\ldots)}_{y}\ \underbrace{=(\ldots01101\ldots)}_{y}\ \underbrace{+(\ldots011100\ldots)}_{z}$$

Contradiction: There is more than one = sign in the resulting expression which means it is not in ADD

Case 2: y has plus sign

$$S = xyz = \underbrace{(\ldots01011\ldots) = (\ldots01101\ldots)}_{x}\ \underbrace{(\ldots101\ldots+\ldots010\ldots)}_{y}\ \underbrace{(\ldots011100\ldots)}_{z}$$

Then xy^2z must also be in ADD

$$\underbrace{(\ldots01011\ldots) = (\ldots01101\ldots)}_{x}\ \underbrace{(\ldots101\ldots+\ldots010\ldots)}_{y}\ \underbrace{(\ldots101\ldots+\ldots010\ldots)}_{y}\ \underbrace{(\ldots011100\ldots)}_{z}$$

Contradiction: There is more then one plus sign in the resulting expression which means it is not in ADD

Case 3: y comes before = sign

$$s = xyz = \underbrace{(\ldots 01011 \ldots 00)}_{x}\ \underbrace{(\ldots 101 \ldots)}_{y}\ \underbrace{=(\ldots 01101 \ldots) + (\ldots 011100 \ldots)}_{z}$$

Then xy^2z must also be in ADD

$$\underbrace{(\ldots 01011 \ldots 00)}_{x}\ \underbrace{(\ldots 0101 \ldots)}_{y}\ \underbrace{(\ldots 101 \ldots)}_{y}\ \underbrace{=(\ldots 01101 \ldots 00) + (\ldots 0111 00 \ldots 00)}_{z}$$

Contradiction: by adding binary before the equal sign and not after we have thrown off the equality of the statement. The left side no longer equals the right side ∴ the resulting expression cannot be in ADD

Case 4: y comes after = sign but before + sign

$$s = xyz = \underbrace{(\ldots 01011 \ldots) =}_{x}\ \underbrace{(\ldots 01101 \ldots)}_{y}\ \underbrace{+(\ldots 0111 00 \ldots)}_{z}$$

Then xy^2z must also be in ADD

$$\underbrace{(\ldots 01011 \ldots) =}_{x}\ \underbrace{(\ldots 01101 \ldots)}_{y}\ \underbrace{(\ldots 01101 \ldots)}_{y}\ \underbrace{+(\ldots 01101 \ldots)}_{z}$$

Contradiction: by adding binary in the space b/t the = and + symbols we change the value of one of the numbers being added on the right side of the expression. This breaks the equality of ADD and thus this expression cannot be a part of ADD.

Case 5: y comes after plus sign

$$s = xyz = \underbrace{(\ldots 01011 \ldots) = (\ldots 01101 \ldots) +}_{x}\ \underbrace{(\ldots 01 \ldots)}_{y}\ \underbrace{(\ldots 0111 00 \ldots)}_{z}$$

Then xy^2z must also be in ADD

$$\underbrace{(\ldots 01011 \ldots) = (\ldots 01101 \ldots) +}_{x}\ \underbrace{(\ldots 01 \ldots)}_{y}\ \underbrace{(\ldots 01 \ldots)}_{y}\ \underbrace{(\ldots 0111 000 \ldots)}_{z}$$

Contradiction: by adding binary in the space after the + symbol we change one of the values being summed on the right side of the expression. This will mess with the overall equality of the statement since nothing on the left side of the equation was added/altered. Thus this expression cannot be in ADD.

Conclusion:
In all possible cases of placing y in S, there is a contradiction to ADD being regular. ADD does not have the property described in the pumping lemma and ADD is not regular □

Bonus Problem:

Regular expressions apply to computer science tasks because they can be used to find substrings. Regular expressions can help us find substrings in programs. Substrings are used in things like control F that highlights where that substring appears on a website or pdf. Regular expressions can also be helpful for passwords. Passwords often have rules such as having a special character, numbers, or letters of different casing. A regular expression can be used to determine if those conditions are met while not caring about what the other characters are in the string. Regular expressions can also be used to clean up input or alter strings w/ find and replace functions. They can search the string for specific characters/groups of characters and replace them with the desired one. This can be generalized out to regular languages. RE's recognize regular languages just like NFA and DFA machines. Regular expressions and regular languages can be used to find patterns or validate input. RE's and regular languages can be used to find patterns such as wanting to make sure all strings accepted to a function start and end with the same letter or are of a certain length. This kind of pattern recognition would help in insuring that input to functions followed the rules of the function for strings. They can be used in password validation because you have a set pattern or order of characters the machine must see to authorize a login.