University of Victoria

CSC 320 - Spring 2023

Foundations of Computer Science

Tutorial 03

Teaching Team

Learning Outcomes:

- Design a regular expression for a language.
- Convert a regular expression to an NFA.
- Convert a DFA to a regular language.

Interesting Article:

"Compressing Regular Expressions' DFA Table by Matrix Decomposition" [1]

January 31st, 2023

Question 01

Design a regular expression for the following languages over...

$$\Sigma = \{0, 1\}$$ Note: $\{0,1\} = (0 \cup 1)$

(a) $L_1 = \{w \mid$ **every odd position of** w **is a** $1\}$

We can start by determining what is and isn't included.

NOT: 0, ε, 000, 00, etc.

IN: 1, 111, 101, 10, etc.

So, we can observe the following pattern:

1, 1$\{0,1\}$, 1 $\{0,1\}$ 1, 1 $\{0,1\}$ 1 $\{0,1\}$, etc.

$$\boxed{R = (1(0 \cup 1))^* \cup (1(0 \cup 1))^* 1}$$

Note: Remember to formalize your answer, don't forget!

(b) $L_2 = \{w \mid w$ **is a string of length at most** $5\}$

We want the option of having 0, 1, 2, 3, 4, and 5 "characters" in our string w. So, we will need ε to be an option.

$$\boxed{(\varepsilon \cup \Sigma)(\varepsilon \cup \Sigma)(\varepsilon \cup \Sigma)(\varepsilon \cup \Sigma)(\varepsilon \cup \Sigma) = R}$$

Remember: $\{0,1\} = \Sigma$

We could also view it as $(\varepsilon \cup \Sigma)^n$ where $n \geq 0$ and $n \leq 5$.

(c) $L_3 = \{w \mid w$ **contains an even number of** 0**'s or exactly two** 1**'s**$\}$

Since we have an or we will use union to create our regular expressions.

We will begin with

— an even number of 0's:

NOT: 10, 0, 110, 011,

IN: 1, 11, 111, 00, 1010, 10101,

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

We can any number of 1's, but can only have an even number of 0's so each 0 must have another.

Note: Here we observe zero 0's as an even number of 0's.

— exactly two 1's:

NOT: 1, 111, 01, 1011,

IN: 11, 110, 101, 011,

$(0^* 1 0^* 1 0^*)$

We can have any number of 0's in between our 1's.

$$\boxed{R = 1^* \cup (1^* 0 1^* 0 1^*)^* \cup (0^* 1 0^* 1 0^*)}$$

Question 02

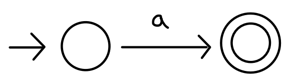
Convert the following regular expression to an NFA...

$$R_1 = (a \cup b^*)a$$

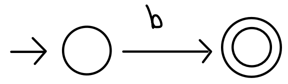
Note: Th formalization of th regular expression.

Step 1

We can draw an NFA that accepts a single a:

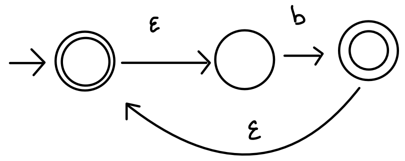


Next we can draw an NFA that accepts a single b:

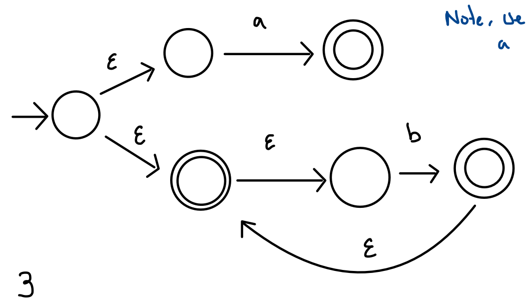


Step 2

An NFA that accepts 0 or more b's:



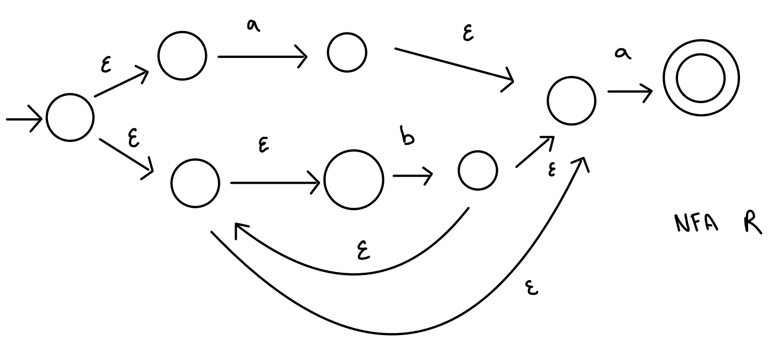
And an NFA that accepts $a \cup b^*$:



Note, we can have a single a or 0 or more b's. but not a's and b's in th same string.

Step 3

Combine our a and $a \cup b^*$ (i.e., $(a \cup b^*)a$)

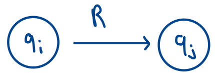


Note: We remove th accept states and now only have a single accept state.

DFA to Regular Expression

If a language is regular, then there exists some regular expression that describes it...

Remember



$\delta_G(q_i, R) = q_j$

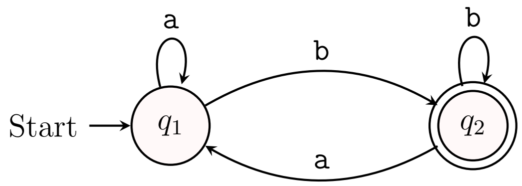
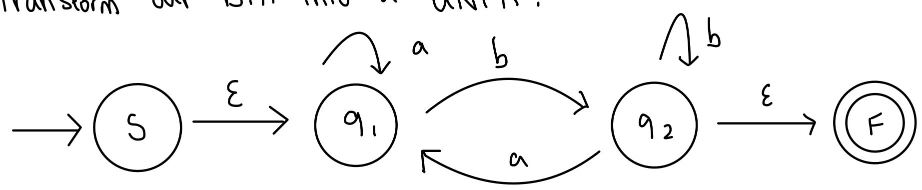


Figure 1: DFA

Step 1

Transform our DFA into a GNFA:



Step 2

We begin by "ripping" out q_2.

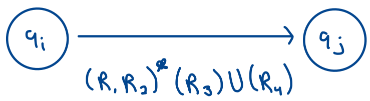
Note: When removing q_{rip} we preserve all regular expressions.

Transitions

q_1, q_2, F $\qquad$ q_1, q_2, q_1

q_1, q_2, q_1, q_2, F $\qquad$ q_2, q_2

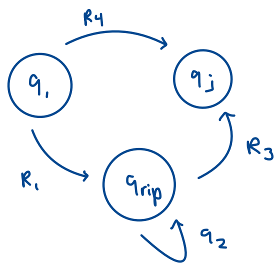
Remember



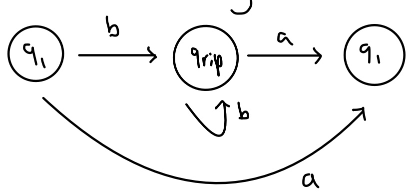
and

$R^+ : RR^*$, $R \cup \emptyset = R$, $R_\varepsilon = R$

and

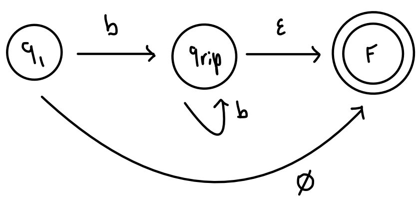


We can do the following:



$R_1 R_{rip} R_3 \cup R_4 = bb^* a \cup a$

Next:



$= bb^+ \varepsilon \cup \emptyset$

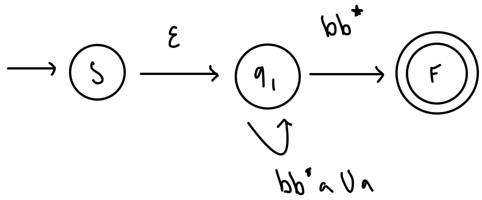
$= bb^*$

Note: Simplify when possible!

Step 3

We can now observe the following:

q_1 to q_1 will be a self loop and we "gain" the transition q_1 to F.



Step 4

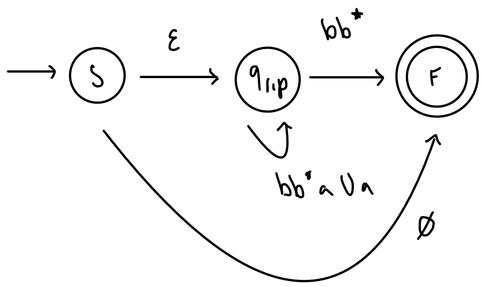
We now can "rip" out state q_1.

__Transitions__

q_1, q_1 $\quad$ q_1, q_1, F

q_1, F

We can do the following:

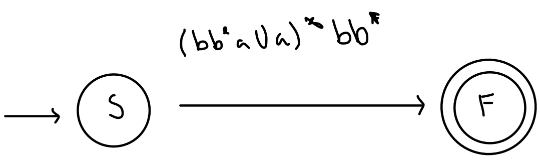


$$= \varepsilon(bb^*a \cup a)^* bb^* \cup \emptyset$$

$$= (bb^*a \cup a)^* bb^*$$

Step 5

Thus we end up with a regular expression as follows:



So we now have some language regular described as some regular expression by definition.

Remember our concluding sentences!

Resources

[1] Y. Liu, L. Guo, P. Liu, and J. Tan, "Compressing regular expressions' dfa table by matrix decomposition," English, in *Implementation and Application of Automata*, ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 282–289, ISBN: 3642180973.