Automata Homework 2

Ali Abbasi – 98105879

December 22, 2022

Contents

1 Description of Automata and Concepts of Regular Languages

1.1

A

We want to design an automaton that recognizes the language of strings over $\{0, 1\}$ that their binary representation's remainder in division by 5 is 2.
Assume that we have an automaton that string s ends up in the state k, if remainder of the binary representation of s in division by 5 is k. Now we derive the transition function of this automaton. Assume that after reading the string s, the next character is a. So our new string is $s \circ a$. We can calculate the remainder of this new string divided by 5 as follows:

$$\begin{aligned} s \circ a \mod 5 &= s \circ 0 + a \mod 5 && (1.1.1)\\ &= 2s + a \mod 5 && (1.1.2)\\ &= 2k + a \mod 5 && (1.1.3)\\ & && (1.1.4) \end{aligned}$$

Where k is the remainder of s in division by 5. So we have:

$$\delta(k, a) = 2k + a \mod 5 \tag{1.1.5}$$

The DFA for this automaton is shown in Figure 1.

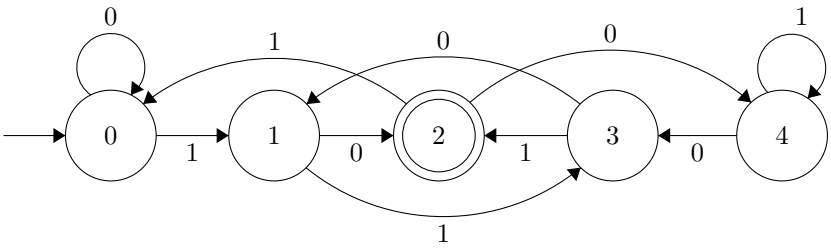


Figure 1: DFA describing the language.

B

First we design the NFA that recognizes the language of all strings with minimum size of 4 over the alphabet $\{a, b\}$ that their first and second to last letters are different. The NFA is shown in Figure 2.

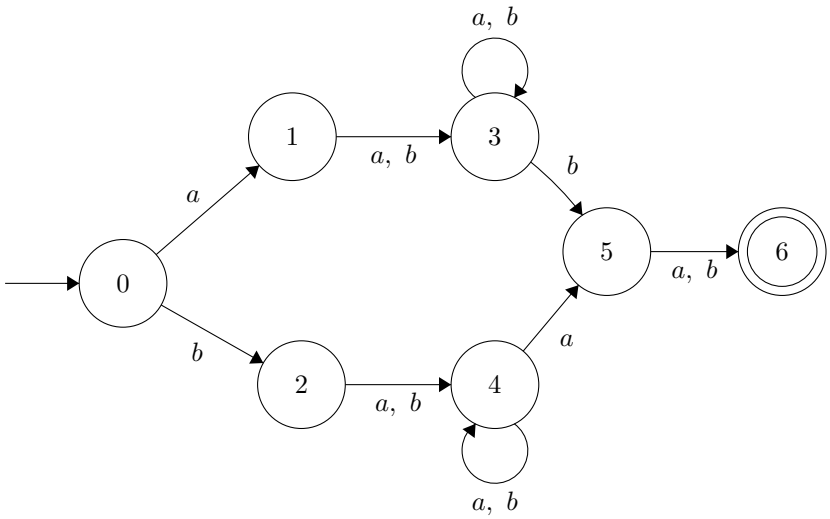


Figure 2: NFA describing the language.

The NFA first gets a or b and gets one or more arbitrary characters. When it reaches to the last two characters, it expects to get a different character than the first two characters. Then it gets another arbitrary character and accept state is reached. Note that state 1 and 2 are introduced only to make the minimum length of strings in this language 4.
Then we convert the NFA to an equivalent DFA using the subset construction algorithm. The DFA is shown in Figure 3.

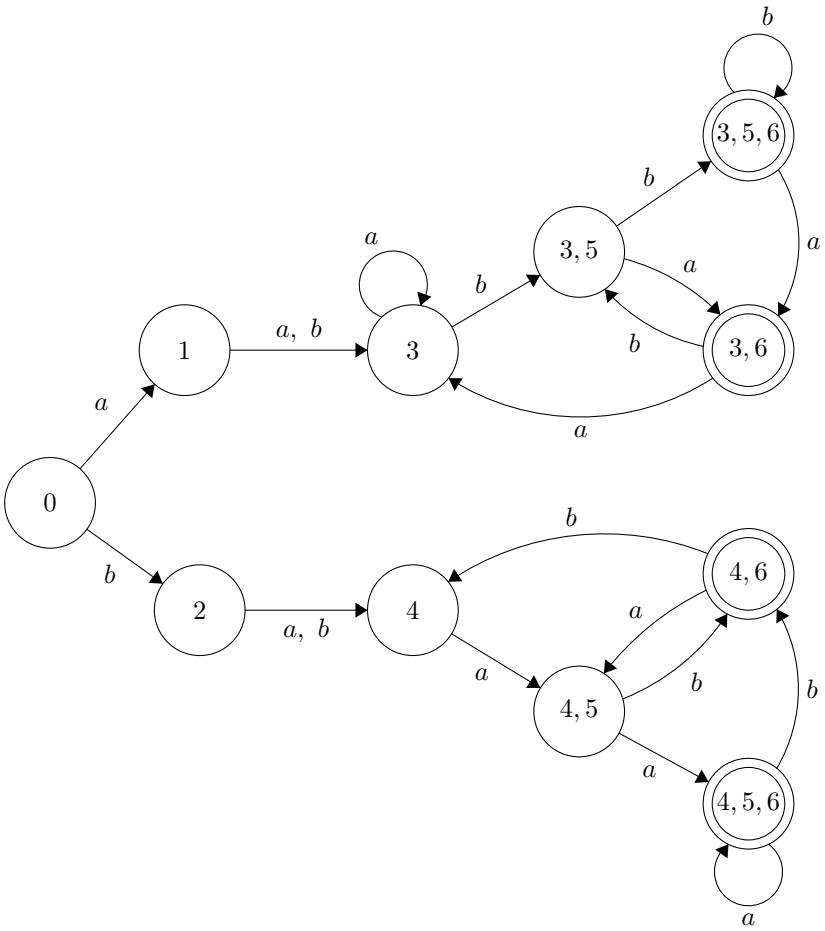


Figure 3: The Equivalent DFA describing the language.

1.2

D possibly has multiple accept states. DFA D' will have the same structure as D, and accept states that are subset of the those of D. Think of DFA D as a graph with states as nodes and transitions as edges. If an accept state in D (for example f_1), has a path to another accept state (f_2), then for all w accepted via f_1, there is a non-empty string x such that wx is accepted via f_2. In this case we remove f_1 from the accept states of D'. Using DFAs notation, we have:

$$D = (Q, \Sigma, \delta, q_0, F) \tag{1.2.1}$$

$$\implies D' = (Q, \Sigma, \delta, q_0, F') \tag{1.2.2}$$

$$\text{where } F' = \{f \in F | \forall f' \in F, \text{ there is no path from } f \text{ to } f'\} \tag{1.2.3}$$

1.3

A

$$\left.\begin{aligned} &g \circ f(q_{0,1}) = g(q_{0,2}) = q_{0,3} \\ &g \circ f(\delta_1(q,a)) = g(\delta_2(f(q),a)) = \delta_3(g \circ f(q),a) \end{aligned}\right\} \implies g \circ f : Q_1 \to Q_3 \text{ is a morphism.} \quad (1.3.1)$$

B

We prove this by induction on the length of the string. Suppose $w = w_1 w_2 \cdots w_n$. We have:
Base Case: $n = 1$:

$$h(\delta_1(q, w_1)) = \delta_2(h(q), w_1) \quad (1.3.2)$$

Induction Hypothesis: Suppose it holds for $n = k$. Then:

$$w_{1:k} \triangleq w_1 w_2 \cdots w_k \quad (1.3.3)$$
$$h(\delta_1^*(q, w_{1:k})) = \delta_2^*(h(q), w_{1:k}) \quad (1.3.4)$$

Inductive Step:

$$\begin{aligned} h(\delta_1^*(q, w_{1:k+1})) &= h(\delta_1(\delta_1^*(q, w_{1:k}), w_{k+1})) && (1.3.5)\\ &= \delta_2(h(\delta_1^*(q, w_{1:k})), w_{k+1}) && (1.3.6)\\ &= \delta_2(\delta_2^*(h(q), w_{1:k}), w_{k+1}) \qquad \text{(Induction Hypothesis)} && (1.3.7)\\ &= \delta_2^*(h(q), w_{1:k+1}) && (1.3.8) \end{aligned}$$

So it holds for all $w \in \Sigma^*$ with length $n > 0$ (Note that $\delta(q, \epsilon)$ is undefined for DFA).

2 Equivalence and Minimization

2.1

q	$\delta(q,a)$	$\delta(q,b)$
01	234	$\emptyset$
234	014	0134
014	234	$\emptyset$
0134	01234	014
01234	01234	0134
$\emptyset$	$\emptyset$	$\emptyset$

Table 1: States and transition functions of the equivalent DFA

Each state of the DFA is a subset of NFA's states. And if it contains the accept state of NFA (4), it is an accept state of DFA. Introducing new names for brevity:

q	$\delta(q,a)$	$\delta(q,b)$
A	B	F
B	C	D
C	B	F
D	E	C
E	E	D
F	F	F

Table 2: States and transition functions of the DFA

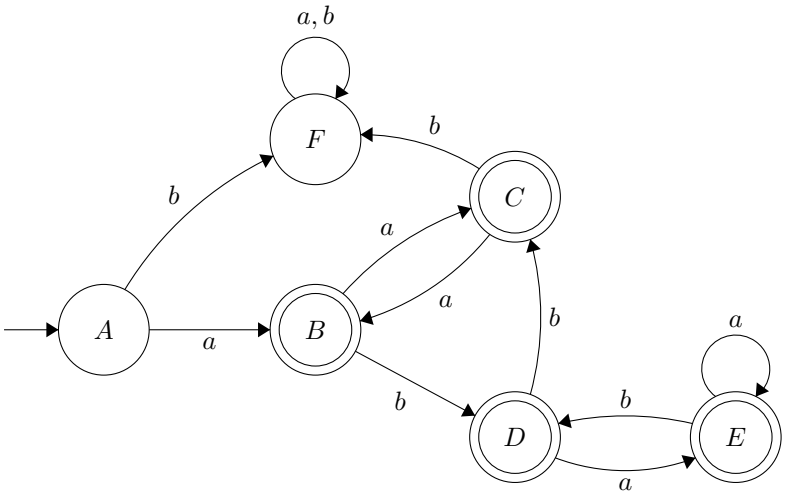


Figure 4: The Equivalent DFA.

2.2

Note that both accept states of DFA are unreachable! So we can remove them and the transitions to them. And the Hopcroft algorithm ends in the first step, since all remaining states are in one class. So the minimized DFA is a single non-accept state.

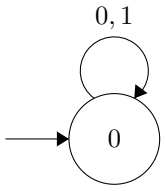


Figure 5: The minimized DFA!

3 Regular Expressions and Grammars

3.1

A

First we design DFA describing $(a \cup b)^*$ language using the rules we learned. Then we simplify it with removing extra ϵ transitions (Figure 6).

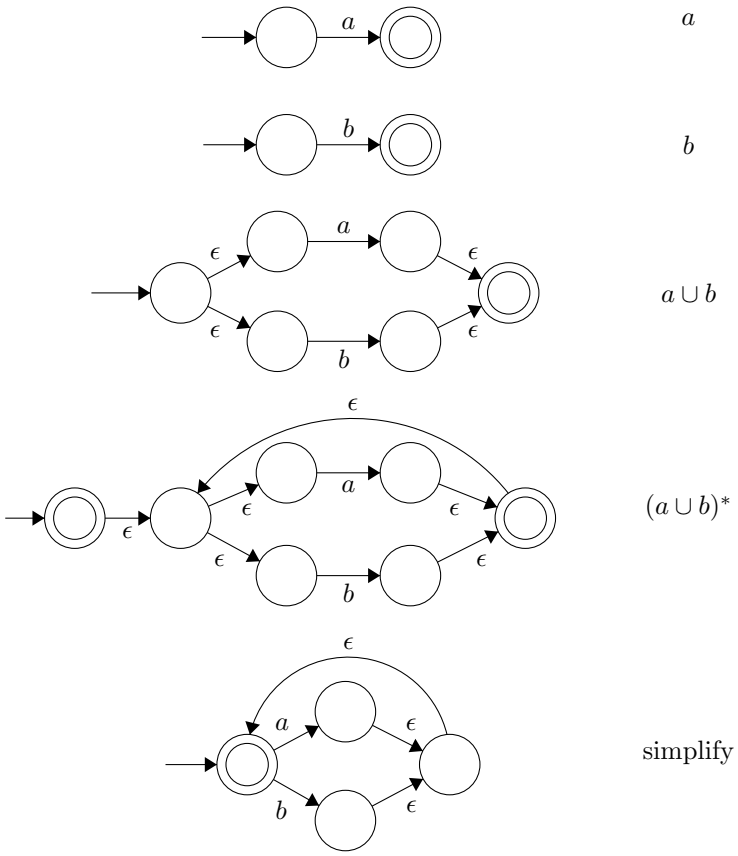


Figure 6: DFA for $(a \cup b)^*$.

Now we design DFA for $(a \cup b)^*(a \cup ba)^*a(b \cup ab)^*$ in the same way (Figure 7).

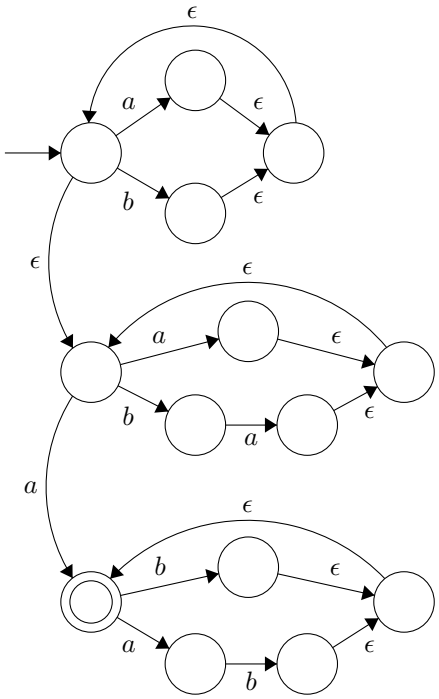


Figure 7: DFA for $(a \cup b)^*(a \cup ba)^*a(b \cup ab)^*$.

B

Using the same method as last part, we derive the following DFA for this language (Figure 8).

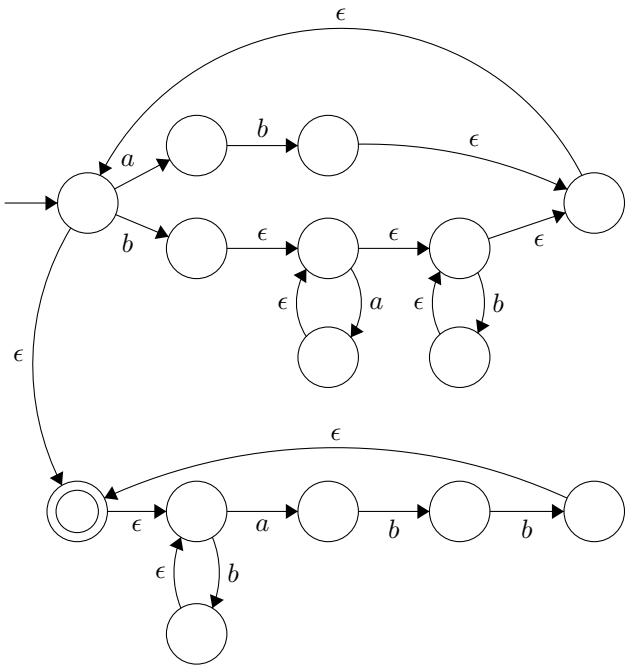
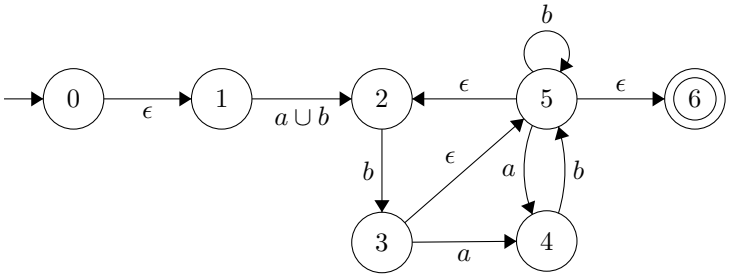


Figure 8: DFA for $(ab \cup ba^*b^*)^*(b^*abb)^*$.

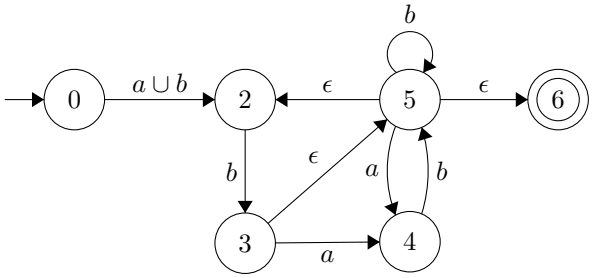
3.2

A

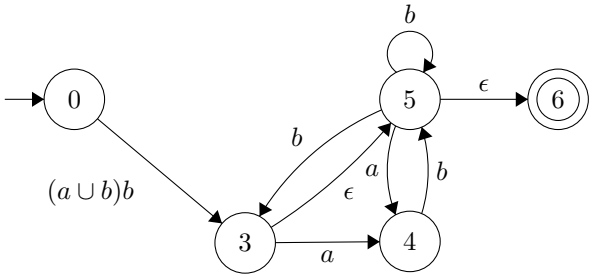
First we convert the DFA to GNFA. Then we rip the GNFA states one by one, to get the regular expression (Figure 9).
Note that to avoid overcrowding the figure, we do not draw the arrows with $\emptyset$ labels.



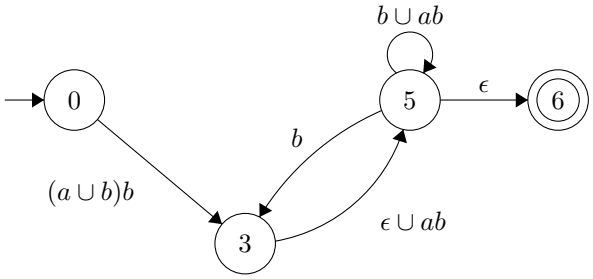
(a) The initial GNFA.



(b) After ripping 1.

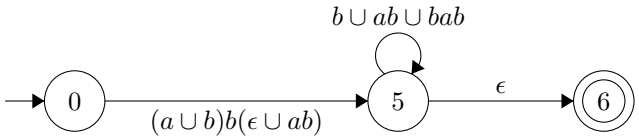


(c) After ripping 2.

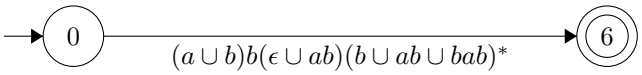


(d) After ripping 4.

Figure 9: Converting DFA to Regular Expression.



(e) After ripping 3 ($b \cup ab \cup b(\epsilon \cup ab) = b \cup ab \cup bab$).



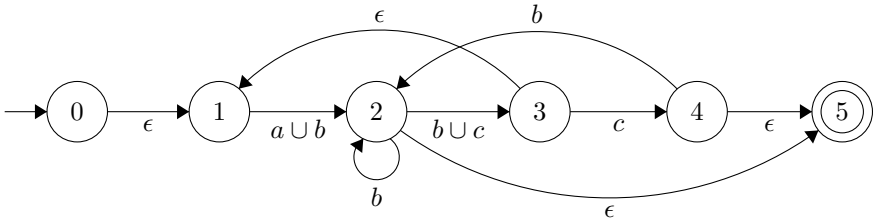
(f) After ripping 5.

Figure 9: Converting DFA to Regular Expression. (Continued)

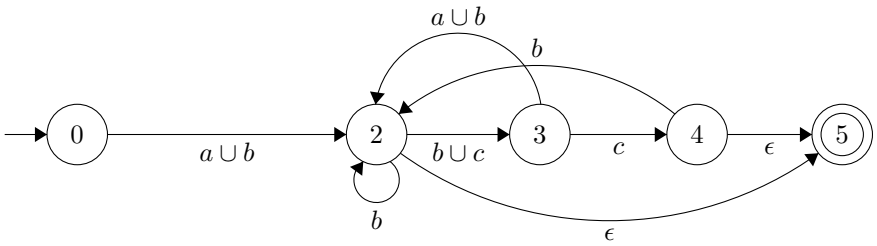
So the resulting regular expression is $(a \cup b)b(\epsilon \cup ab)(b \cup ab \cup bab)^*$.

B

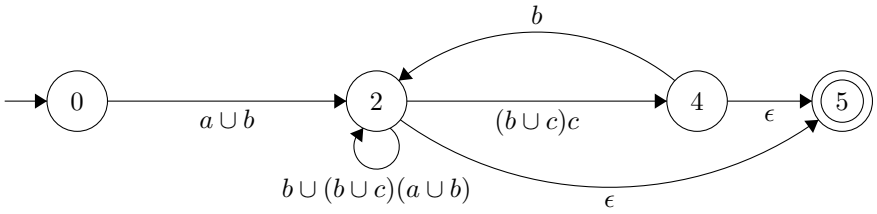
Using the same procedure as last part, we can get the corresponding regular expression (Figure 10).



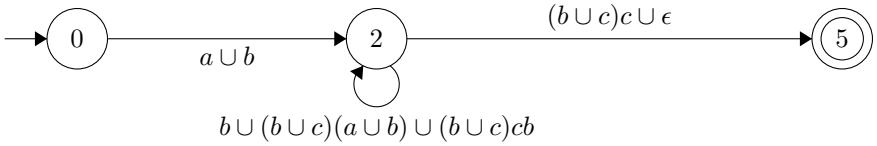
(a) The initial GNFA.



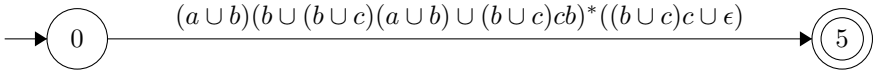
(b) After ripping 1.



(c) After ripping 3.



(d) After ripping 4.



(e) After ripping 2.

Figure 10: Converting DFA to Regular Expression.

So the resulting regular expression is $(a \cup b)(b \cup (b \cup c)(a \cup b) \cup (b \cup c)cb)^*((b \cup c)c \cup \epsilon)$.

3.3

A

We know:

$$\left.\begin{array}{l} x \in x^* \\ y \in yx^* \end{array}\right\} \implies (x \cup y) \subseteq (x^* \cup yx^*) \implies (x \cup y)^* \subseteq (x^* \cup yx^*)^* \tag{3.3.1}$$

and also we know that $(x \cup y)^*$ is language of all strings of arbitrary length over the alphabet $\{a, b\}$. So:

$$(x^* \cup yx^*)^* \subseteq (x \cup y)^* \tag{3.3.2}$$

Thus we have:

$$(x \cup y)^* = (x^* \cup yx^*)^* \tag{3.3.3}$$

B

We use result of the last part:

$$\begin{aligned}
(y^*(x \cup \epsilon)y^*)^* &= (y^*xy^* \cup y^*y^*)^* && (3.3.4)\\
&= (y^*xy^* \cup y^*)^* && (3.3.5)\\
&= (y^* \cup y^*xy^*)^* && (3.3.6)\\
\xrightarrow{b=y, a=y^*x} &= (b^* \cup ab^*)^* && (3.3.7)\\
&= (b \cup a)^* && \text{using the last part} \quad (3.3.8)\\
&= (y \cup y^*x)^* && (3.3.9)
\end{aligned}$$

Again using the same reasoning as last part, we can get:

$$\left.\begin{array}{l} x \in y^*x \implies (y \cup x) \subseteq (y \cup y^*x) \implies (y \cup x)^* \subseteq (y \cup y^*x)^* \\ (y \cup y^*x)^* \subseteq \text{language of all strings over } \{a, b\} \implies (y \cup x)^* \subseteq (y \cup x)^* \end{array}\right\} \tag{3.3.10}$$

$$\implies (y^*(x \cup \epsilon)y^*)^* = (y \cup y^*x)^* = (y \cup x)^* = (x \cup y)^* \tag{3.3.11}$$

3.4

A

We know that A generates strings of arbitrary length containing a (i.e. a^*). So strings generated by S are of the form $a^*(a^*ba^*b)^*b$. And language of this regex can be described by the following NFA (Figure 11).

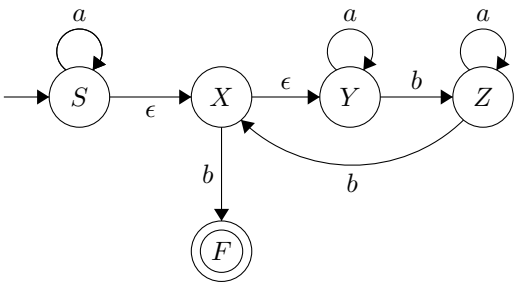


Figure 11: DFA for the regex $a^*(a^*ba^*b)^*b$.

And we can convert this NFA to a left-linear grammar easily.

$$\begin{aligned} S &\to aS \mid X \\ X &\to Y \mid b \\ Y &\to aY \mid bZ \\ Z &\to aZ \mid bX \end{aligned} \tag{3.4.1}$$

B

If we examine this grammar carefully, we can see that it describes language of strings with even number of as (because either $S \Rightarrow B$ and S has no as, or $S \Rightarrow AA$, where each time using $A \to AAA$ rule, adds 2 as to the string). So we can design the following DFA (Figure 12) to recognize this language.

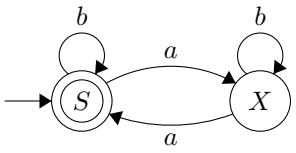


Figure 12: DFA for the language of strings with even number of as.

The equivalent left-linear grammar is:

$$\begin{aligned} S &\to bS \mid aX \mid \epsilon \\ X &\to bX \mid aS \end{aligned} \tag{3.4.2}$$

4 Closure Properties of Regular Languages

4.1

We design a NFA which recognizes the language of $\text{put}(L, x)$ (i.e., D_2) using the NFA of L (i.e., D). The main idea is to duplicate every state and transition of D and add a new transition from every state of D to their corresponding duplicate state, accepting the input x (for brevity, here we assume that our new transitions can input a string of arbitrary length, instead of a single character). And

finally, only the duplicate of the final state of D will be accepting states. Doing so, we the read inputs and proceed in the initial NFA, D. As soon as x is read, we switch to the duplicate of the current state and proceed in the duplicate NFA until we reach its final state. We can show this procedure formally as follows:

$$D = (Q, \Sigma, \delta, q_0, F) \tag{4.1.1}$$

We denote the corresponding state of q as q'.

$$D_2 = (Q_2, \Sigma, \delta_2, q_0, F') \tag{4.1.2}$$

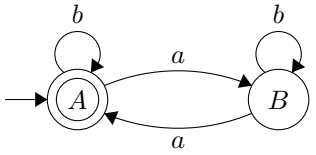
where:

$$Q' \triangleq \bigcup_{q \in Q} \{q'\} \quad \text{and} \quad F' \triangleq \bigcup_{q \in F} \{q'\} \tag{4.1.3}$$

$$Q_2 = Q \cup Q' \tag{4.1.4}$$

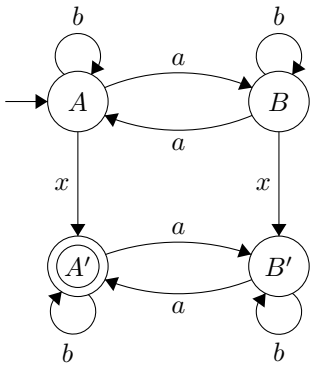
$$\delta_2(s,a) = \begin{cases} \delta(q,a) & \text{if } s = q \in Q \text{ and } a \neq x \\ \delta(q,a) \cup \{q'\} & \text{if } s = q \in Q \text{ and } a = x \\ \delta'(q,a) & \text{if } s = q' \in Q' \end{cases} \tag{4.1.5}$$

Clarifying using an example: $L =$ all strings over $\{a, b\}$ with even number of as.



(a) D.

$\Longrightarrow$



(b) D_2.

And D_2 describes $\text{put}(L, x)$.

4.2

We will first solve part B.

B

We want to prove that if L is a regular language, $\text{SQRT}(L) = \left\{x \mid \exists y : \ |y| = |x|^2, xy \in L\right\}$ is a regular language.
Assume the DFA recognizing L is $D = (Q, \Sigma, \delta, q_0, F)$.

We define $\mathcal{G}$ to be the set of all functions from Q to $\mathcal{P}(Q)$ (power-set of Q). And for any $f, g \in \mathcal{G}$, we define $f \circ g$ as follows:

$$f \circ g(q) = \bigcup_{q' \in g(q)} f(q') \tag{4.2.1}$$

Construct the new DFA D' recognizing SQRT(L) as follows:

$$D' = (Q', \Sigma, \delta', q_0', F') \tag{4.2.2}$$
$$Q' = Q \times \mathcal{G}^2 \tag{4.2.3}$$
$$\delta'((q, g_1, g_2), a) = \Big(\delta(q, a), g_1 \circ g_0, ((g_2 \circ g_1) \circ g_1) \circ g_0\Big) \tag{4.2.4}$$
$$\text{Where } g_0(q) = \{\delta(q, a) \mid a \in \Sigma\} \tag{4.2.5}$$
$$q_0' = (q_0, I, I) \tag{4.2.6}$$
$$\text{Where } I(q) = \{q\} \tag{4.2.7}$$
$$F' = \big\{(q, g_1, g_2) \in Q \times \mathcal{G}^2 \mid g_2(q) \cap F \neq \emptyset\big\} \tag{4.2.8}$$

In this new DFA, after reading a word w, if D reaches state q, then D' reaches the state (q, g_1, g_2) where $g_1(.)$ and $g_2(.)$ represent the set of states of D that can be reached starting at . then reading a string of length $|w|$ and $|w|^2$ respectively. This can be proved using induction on strings of length n. When $n = 0$ (no character is read), $g_1(.)$ and $g_2(.)$ are contain only . state.
If our claim holds for strings of length $n = k$, we have:

$$\delta'(q_0', w) = (q, g_1, g_2) \qquad \forall w : |w| = k \tag{4.2.9}$$
$$g_1(.) = \{\text{all states reachable from . with } k \text{ steps}\} \tag{4.2.10}$$
$$g_2(.) = \big\{\text{all states reachable from . with } k^2 \text{ steps}\big\} \tag{4.2.11}$$
$$g_0(.) = \{\delta(., a) \mid a \in \Sigma\} \tag{4.2.12}$$
$$\implies g_0(.) = \{\text{all states reachable from . with 1 step}\} \tag{4.2.13}$$

Now for $n = k + 1$, we have:

$$\forall a \in \Sigma : \delta'((q, g_1, g_2), a) = \Big(\delta(q, a), g_1 \circ g_0, ((g_2 \circ g_1) \circ g_1) \circ g_0\Big) \tag{4.2.14}$$
$$\implies g_1 \circ g_0(.) = \{\text{all states reachable from . with } k + 1 \text{ steps}\} \tag{4.2.15}$$
$$\text{and } ((g_2 \circ g_1) \circ g_1) \circ g_0 = \{\text{all states reachable from . with } \underbrace{k^2 + k + k + 1}_{=(k+1)^2} \text{ steps}\} \tag{4.2.16}$$

Thus we have proved our claim. And we have designed a DFA D' recognizing SQRT(L). So SQRT(L) is a regular language.

A

A similar argument can be used to prove this part. This time we use functions in $\mathcal{G}$ to keep track of length of strings in B.

$$A, B \text{ are regular } \implies \mathrm{C}(A, B) = \big\{x \in A \mid \exists y : \ |y| = |x|^2, y \in B\big\} \text{ is regular} \tag{4.2.17}$$

There exists DFAs D_A and D_B recognizing A and B respectively. We design a DFA D_C recognizing $\mathrm{C}(A, B)$ as follows:

$$D_A = (Q_A, \Sigma, \delta_A, q_{0,A}, F_A) \tag{4.2.18}$$
$$D_B = (Q_B, \Sigma, \delta_B, q_{0,B}, F_B) \tag{4.2.19}$$
$$D_C = (Q_C, \Sigma, \delta_C, q_{0,C}, F_C) \tag{4.2.20}$$
$$\text{Where: } Q_C = Q_A \times \mathcal{G}^2 \tag{4.2.21}$$
$$\delta_C((q, g_1, g_2), a) = \Big(\delta_A(q, a), g_1 \circ g_0, ((g_2 \circ g_1) \circ g_1) \circ g_0\Big) \tag{4.2.22}$$
$$\text{Where } g_0(.) = \{\delta_B(q, .) \mid a \in \Sigma\} \tag{4.2.23}$$
$$q_{0,C} = (q_{0,A}, I, I) \tag{4.2.24}$$
$$\text{Where } I(.) = \{.\} \tag{4.2.25}$$
$$F_C = \big\{(q, g_1, g_2) \in Q_A \times \mathcal{G}^2 \mid q \in F_A \text{ and } g_2(q_{0,B}) \cap F_B \neq \emptyset\big\} \tag{4.2.26}$$

5 Proving Non-Regularity of Languages

5.1

A

Proof by contradiction. Assume L_1 is a regular language and its pumping length is p. Consider the string $w = a^p b^{p!+p} c^{p!+p}$. Because w is a member of L_1 and has length more than p, the pumping lemma guarantees that w can be split into three pieces, $w = xyz$, satisfying the lemma conditions. We know that $|xy| \leq p$ and $|y| \geq 1$. So x and y contain only as. Pumping this strings we get:

$$xy^i z \in L_1 \quad \text{for all } i \in \mathbb{N}. \tag{5.1.1}$$

If we take y's length to be k, number of as in $xy^i z$ will be $p - k + ki$. And we can find an i such that the number of as, bs, and cs are equal in $xy^i z$.

$$p - k + ki = p! + p \tag{5.1.2}$$
$$\implies k(i-1) = p! \tag{5.1.3}$$
$$\xrightarrow{k \leq p \implies k \mid p!} i = \frac{p!}{k} + 1 \tag{5.1.4}$$

So the string $a^{p!+p} b^{p!+p} c^{p!+p}$ can be achieved by pumping $a^p b^{p!+p} c^{p!+p}$, but it is not a member of L_1. So we have reached a contradiction and L_1 is not a regular language.

B

Assume L_2 is a regular language, p is its pumping length and r is the smallest prime number such that $p \leq r$. Consider the string a^{2r}. The pumping lemma guarantees that this string can be split into three pieces, $a^{2r} = xyz$, satisfying the lemma conditions. Assuming y's length to be k, we have:

$$|xy^i z| = 2r - k + ik = 2r + (i-1)k \text{ is product of two prime numbers.} \tag{5.1.5}$$

For odd values of i we have:

$$i = 2j + 1 \tag{5.1.6}$$

$$\implies 2r + (2j)k = 2(r + jk) \text{ is product of two prime numbers.} \tag{5.1.7}$$

$$\implies r + jk \text{ is a prime number.} \tag{5.1.8}$$

So we have a linear pattern for generating prime numbers larger than r! And we know that there is no such pattern. So we have reached a contradiction and L_2 is not a regular language.

5.2 To Do