

Pattern Matching

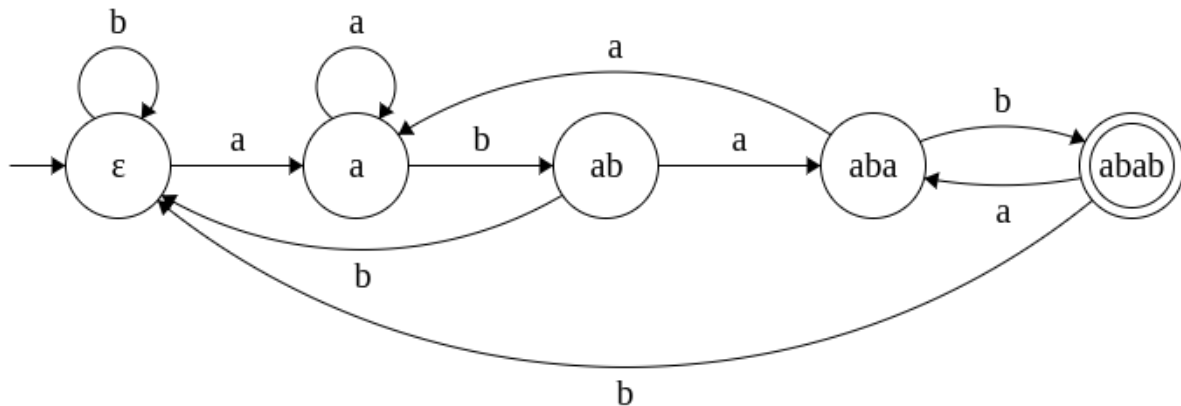
Another prominent application of DFA is in text processing. We discuss three application scenarios:

1. Exact pattern matching
2. Approximate pattern matching
3. Search for multiple patterns

1. Find the number of occurrences of pattern $P = abab$ in text $T = aababababba$.

Given a text T and pattern P , the problem is to find if P appears in T . And, if it appears, how many times it appears? A DFA can be constructed over the pattern and the text T can be run over the DFA. Whenever final state is reached, the repeat count is incremented.

We can design a DFA for the pattern "abab" in the same manner we discussed. The DFA is given below.



Each state is named by part of the pattern that has been observed so far. We initialize a variable *count* to 0. The count will be incremented whenever the final state *abab* is reached.

$$\begin{aligned} \delta(\epsilon, aababababba) &= \delta(a, ababababba) \text{ [count = 0]} \\ &= \delta(a, babababba) \text{ [count = 0]} \\ &= \delta(ab, abababba) \\ &= \delta(aba, bababba) \\ &= \delta(abab, ababba) \text{ [count = 1]} \\ &= \delta(aba, babba) \\ &= \delta(abab, abba) \text{ [count = 2]} \\ &= \delta(aba, bba) \\ &= \delta(abab, ba) \text{ [count = 3]} \\ &= \delta(\epsilon, a) \end{aligned}$$

$$= \delta(a, \epsilon)$$

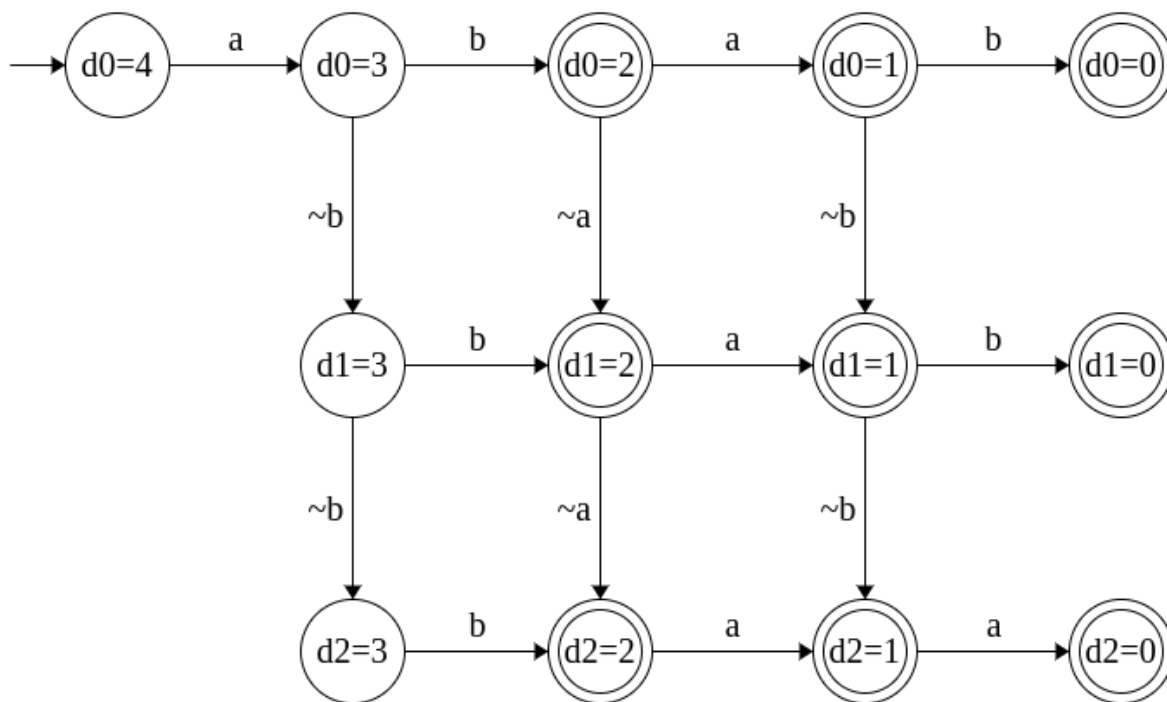
$$= a$$

Thus the number of occurrences can be computed as 3. With additional logic over the DFA, the start positions of the pattern can be computed too.

2. Finding approximate patterns with bounded distances

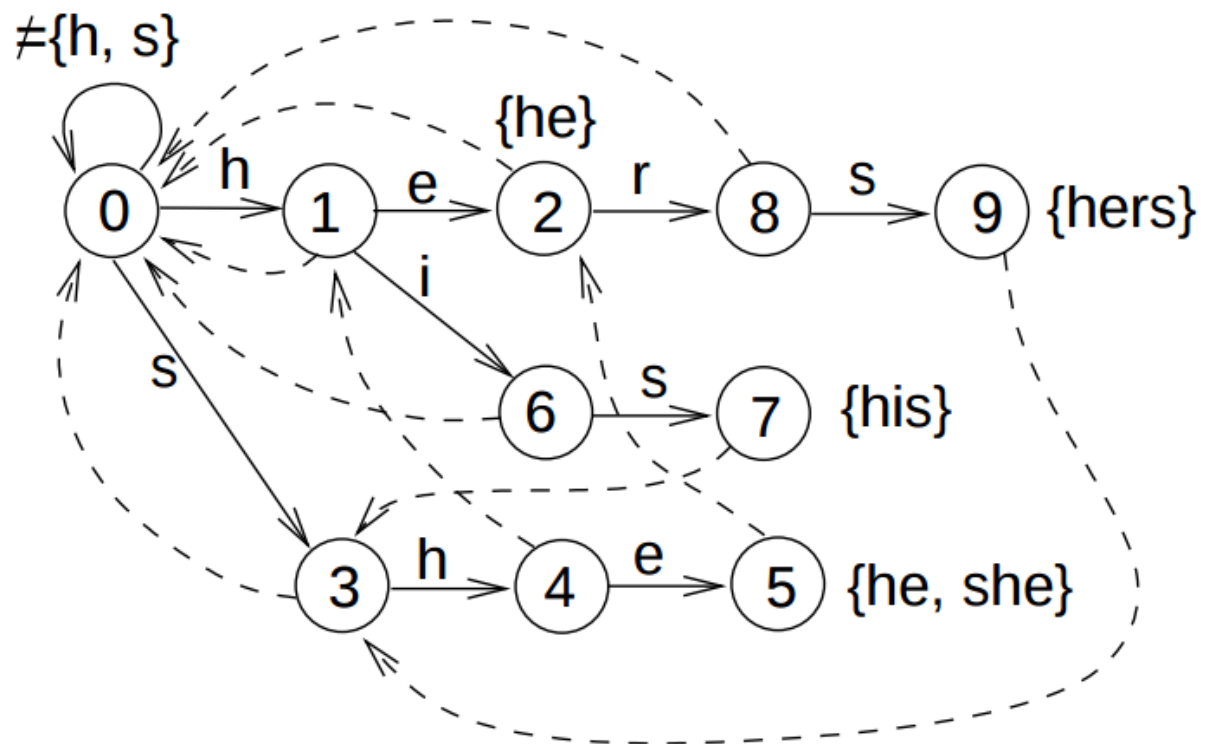
DFA's can also be used to detect approximate patterns or similar strings with bounded distance. As an example, let's consider the same pattern "abab". Assume that we want to identify strings that are closer to it by a distance of 2 at max. Some strings that satisfy the condition are ??abab, ?a?bab, ?ab?ab, ?aba?b, ?abab?, a??bab, a?b?ab, where ? denotes that any character other than one expected is encountered.

The DFA for this scenario is given below. The states are named as follows: $d_i=j$ denotes the fact that the distance differs by j characters with i differences so far. The states with distance $j \leq 2$ are marked as final states.



3. Finding multiple patterns in the same text

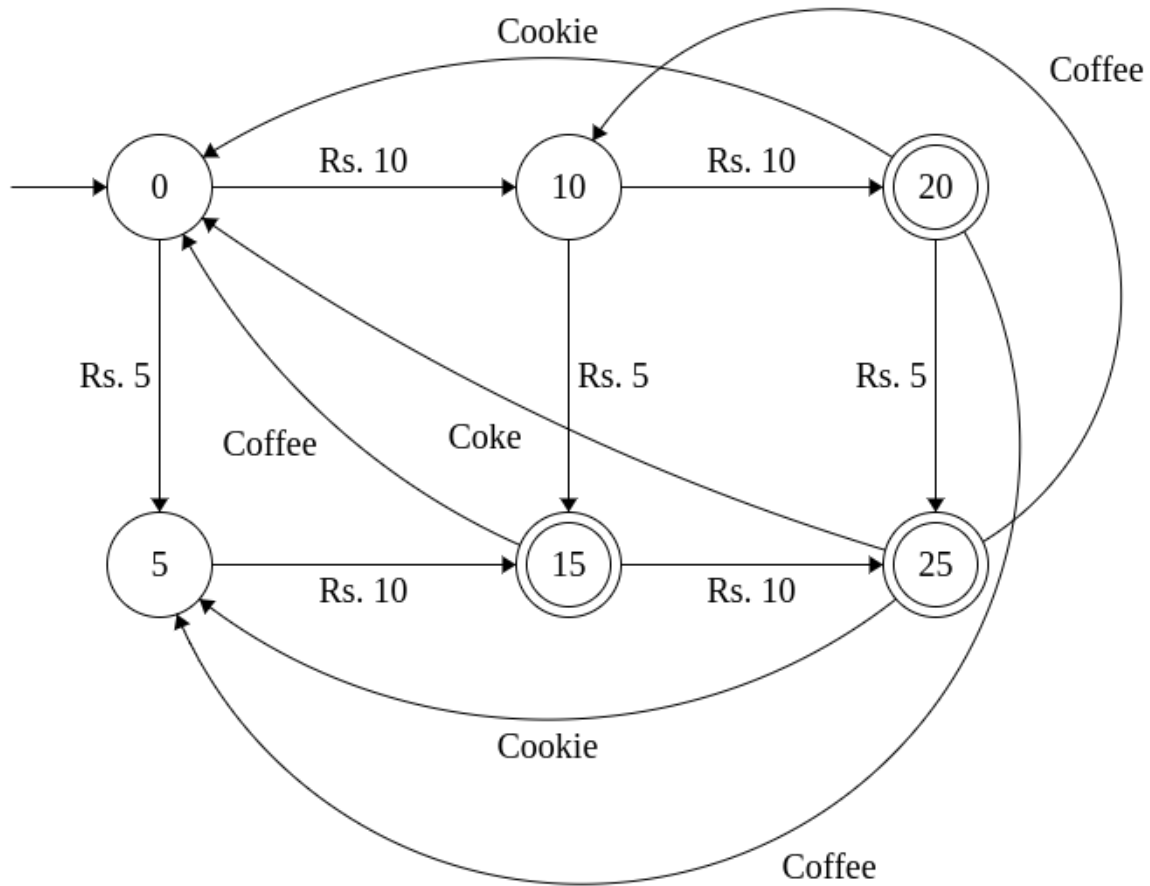
Yet another application of DFA in text processing is finding the occurrences of multiple patterns in a text. Aho-Corasick algorithm provides an approach to construct DFA (referred to as Aho-Corasick automaton), to simultaneously search for multiple patterns. The automaton constructed for the patterns {he, she, his, her} is given below. The image is from Slide 9 of [Set Matching and Aho-Corasick Algorithm by Pekka Kilpelainen](#).



C. Vending Machine

DFA can be used to model real-life scenarios. Here, we see how a vending machine is modeled using DFA. A vending machine accepts inputs in the form of coins or denominations and dispenses items based on the total amount collected and the selection made by the customer

The DFA below is an example of a vending machine that accepts coins of two denominations 5 and 10 and selection of three items Tea, Cookie and Coke that costs Rs. 15, Rs. 20 and Rs. 25 respectively. Although, the state machine could be designed in an elaborate manner, this one gives a fair idea how it can be used to design a real-life scenario.



Regular Grammar

A regular language can also be described using regular grammar which is basically a set of rules using a set of terminals and non-terminals. The terminals are basically the elements of the alphabet Σ and the non-terminals, also referred to as variables, are a composition of terminals and non-terminals. More formally,

Definition: A regular grammar G is a 4-tuple (V, T, P, S) .

- $V \rightarrow$ Variables/Non-terminals that correspond to states Q
- $T \rightarrow$ Terminals that correspond to the alphabet Σ
- $P \rightarrow$ Productions/rules that correspond to transitions δ
- $S \rightarrow$ Start rule that corresponds to the initial state.

Regular expressions and their equivalent grammar

a? \rightarrow Strings with 0 or 1 a. $L = \{\epsilon, a\}$

$S \rightarrow a \mid \epsilon$

a* → Strings with 0 or more a's. $L = \{\epsilon, a, aa, aaa, \dots\}$

$S \rightarrow aS \mid \epsilon$ OR alternately, $S \rightarrow Sa \mid \epsilon$

a⁺ → Strings with 1 or more a's. $L = \{a, aa, aaa, \dots\}$

$S \rightarrow aS \mid a$

a+b → Either a or b. $L = \{a, b\}$

$S \rightarrow a \mid b$

(a+b)(a+b) → Combination of a or b of length 2. $L = \{aa, ab, ba, bb\}$

$S \rightarrow aA \mid bA$

$A \rightarrow a \mid b$

(a+b)* → Strings with any combination of a's and b's. $L = \{\epsilon, a, b, aa, ab, ba, bb, aaa, \dots\}$

$S \rightarrow aS \mid bS \mid \epsilon$

(a+b)*abb → Strings ending with abb. $L = \{abb, aabb, babb, aaabb, ababb, baabb, bbabb, \dots\}$

$S \rightarrow aS \mid bS \mid aB$

$B \rightarrow bA$

$A \rightarrow b$

ab(a+b)* → Strings starting with ab. $L = \{ab, aba, abb, abaa, abab, abba, abbb, \dots\}$

$S \rightarrow aB$

$B \rightarrow bA$

$A \rightarrow aA \mid bA \mid \epsilon$

(a+b)*aa(a+b)* → Strings that contains aa. $L = \{aa, aaa, baa, aab, \dots\}$

$S \rightarrow aS \mid bS \mid aA$

$A \rightarrow aB$

$B \rightarrow aB \mid bB \mid \epsilon$

a*b*c* → 0 or more a's, followed by 0 or more b's, followed by 0 or more c's. $L = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, \dots\}$

$S \rightarrow aS \mid bB \mid cC \mid \epsilon$

$B \rightarrow bB \mid cC \mid \epsilon$

$C \rightarrow cC \mid \epsilon$

$a^+b^+c^+ / aa^*bb^*cc^*$ \rightarrow 1 or more a's, followed by 1 or more b's, followed by 1 or more c's. $L = \{abc, aabc, abbc, abcc, aabbc, aabcc, abbcc, \dots\}$

$S \rightarrow aA$
 $A \rightarrow aA \mid bB$
 $B \rightarrow bB \mid cC$
 $C \rightarrow cC \mid \epsilon$

$(a+b)^*(a+bb)$ \rightarrow Strings that end with a or bb. $L = \{a, bb, aa, abb, ba, bbb, \dots\}$

$S \rightarrow aS \mid bS \mid a \mid bB$
 $B \rightarrow b$

$(aa)^*(bb)^*b$ \rightarrow Strings with even number of a's followed by odd number of b's. $L = \{b, aab, bbb, aabbb, \dots\}$

$S \rightarrow aA \mid bB \mid \epsilon$
 $A \rightarrow aC$
 $C \rightarrow aA \mid bB \mid \epsilon$
 $B \rightarrow bD \mid \epsilon$
 $D \rightarrow bB$

$(0+1)^*000$ \rightarrow Binary strings ending with 3 0's. $L = \{000, 0000, 1000, 00000, 01000, 10000, 11000, \dots\}$

$S \rightarrow 0S \mid 1S \mid 0A$
 $A \rightarrow 0B$
 $B \rightarrow 0$

$(11)^*$ \rightarrow Strings with even number of 1's. $L = \{\epsilon, 11, 1111, 111111, \dots\}$

$S \rightarrow 1A \mid \epsilon$
 $A \rightarrow 1S$

$01^* + 1$ $\rightarrow \{1\} \cup$ Strings that start with 0 followed by zero or more 1's. $L = \{1, 0, 01, 011, \dots\}$

$S \rightarrow 0A \mid 1$
 $A \rightarrow 1A \mid \epsilon$

$(01)^* + 1$ $\rightarrow \{1\} \cup$ Strings with zero or more 01's. $L = \{1, \epsilon, 01, 0101, 010101, \dots\}$

$S \rightarrow 0A \mid 1 \mid \epsilon$
 $A \rightarrow 1B$
 $B \rightarrow 0A \mid \epsilon$

$0(1^* + 1)$ \rightarrow 0 followed by any number of 1's. $L = \{0, 01, 011, 0111, \dots\}$

$S \rightarrow 0A$
 $A \rightarrow 1A \mid \varepsilon$

$(1+\varepsilon)(00^*1)^*0^*$ \rightarrow Strings with no consecutive 1's. $L = \{\varepsilon, 1, 10, 101, 1001, 1010, 10101, 101001, 101010, \dots\}$

$S \rightarrow 1A \mid 0B$
 $A \rightarrow 0B$
 $B \rightarrow 0B \mid 1C$
 $C \rightarrow 0B \mid 0D \mid \varepsilon$
 $D \rightarrow 0D \mid \varepsilon$

Exercises

Hopefully you got the hang now. For the regular expressions given below try to come up with the grammar.

1. a's and b's of length 2

$aa + ab + ba + bb$ OR $(a+b)(a+b)$

2. a's and b's of length ≤ 2

$\varepsilon + a + b + aa + ab + ba + bb$ OR $(\varepsilon + a + b)(\varepsilon + a + b)$ OR $(a+b)?(a+b)?$

3. a's and b's of length ≤ 10

$(\varepsilon + a + b)^{10}$

4. Even-lengthed strings of a's and b's

$(aa + ab + ba + bb)^*$ OR $((a+b)(a+b))^*$

5. Odd-lengthed strings of a's and b's

$(a+b)((a+b)(a+b))^*$

6. $L(R) = \{ w : w \in \{0,1\}^* \text{ with at least three consecutive 0's} \}$

$(0+1)^* 000 (0+1)^*$

7. Strings of 0's and 1's with no two consecutive 0's

$$(1^+ 0 1^*)^* \text{ OR } (11^* 0 1^*)^* \text{ OR } (1 + 01)^* (0 + \varepsilon)$$

8. Strings of a's and b's starting with a and ending with b.

$$a (a+b)^* b$$

9. Strings of a's and b's whose second last symbol is a.

$$(a+b)^* a (a+b)$$

10. Strings of a's and b's whose third last symbol is a and fourth last symbol is b.

$$(a+b)^* b a (a+b) (a+b)$$

11. Strings of a's and b's whose first and last symbols are the same.

$$(a (a+b)^* a) + (b (a+b)^* a)$$

12. Strings of a's and b's whose first and last symbols are different.

$$(a (a+b)^* b) + (b (a+b)^* a)$$

13. Strings of a's and b's whose last and second last symbols are same.

$$(a+b)^* (aa + bb)$$

14. Strings of a's and b's whose length is even or a multiple of 3 or both.

$$R1 + R2 \text{ where } R1 = ((a+b)(a+b))^* \quad \text{and} \quad R2 = ((a+b)(a+b)(a+b))^*$$

15. Strings of a's and b's such that every block of 4 consecutive symbols has at least 2 a's.

$$(aaxx + axax + axxa + xaax + xaxa + xxaa)^* \text{ where } x = (a+b)$$

$$16. L = \{a^n b^m : n \geq 0, m \geq 0\}$$

$$a^* b^*$$

$$17. L = \{a^n b^m : n > 0, m > 0\}$$

$$aa^* bb^* \text{ OR } a^+ b^+$$

$$18. L = \{a^n b^m : n + m \text{ is even}\}$$

$$aa^* bb^* + a(aa)^* b(bb)^*$$

$$19. L = \{a^{2n}b^{2m} : n \geq 0, m \geq 0\}$$

$$(aa)^* (bb)^*$$

20. Strings of a's and b's containing not more than three a's.

$$b^* (\epsilon + a) b^* (\epsilon + a) b^* (\epsilon + a) b^*$$

$$21. L = \{a^n b^m : n \geq 3, m \leq 3\}$$

$$aaa a^* (\epsilon + b) (\epsilon + b) (\epsilon + b)$$

$$22. L = \{ w : |w| \bmod 3 = 0 \text{ and } w \in \{a,b\}^* \}$$

$$((a+b)(a+b)(a+b))^*$$

$$23. L = \{ w : n_a(w) \bmod 3 = 0 \text{ and } w \in \{a,b\}^* \}$$

$$b^* a b^* a b^* a b^*$$

24. Strings of 0's and 1's that do not end with 01

$$(0+1)^* (00 + 10 + 11)$$

$$25. L = \{ uvv : u, v \in \{a,b\}^* \text{ and } |v| = 2 \}$$

$$(aa + ab + ba + bb) (a+b)^* (aa + ab + ba + bb)$$

26. Strings of a's and b's that end with ab or ba.

$$(a+b)^* (ab + ba)$$

$$27. L = \{a^n b^m : m, n \geq 1 \text{ and } mn \geq 3\}$$

$$a bbb b^* + aaa a^* b + aa a^* bb b^*$$

Regular Grammar - NFA Equivalence

In the last section, we saw how regular languages can be specified using regular grammar. The grammar was written in a particular fashion. For instance,

We wrote the grammar for $(a+b)(a+b)$ as $S \rightarrow aA \mid bA$; $A \rightarrow a \mid b$. This is not the only way to write the grammar. We could write as $S \rightarrow aa \mid ab \mid ba \mid bb$ which is equally correct.

Similarly, we could have written the grammar for $(a+b)^*abb$ as $S \rightarrow aS \mid bS \mid abb$. But we chose to write it as $S \rightarrow aS \mid bS \mid aB$; $B \rightarrow bC$; $C \rightarrow b$.

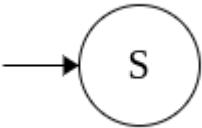
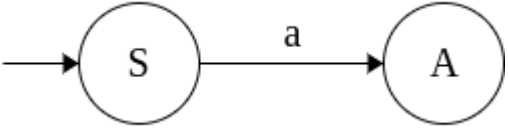
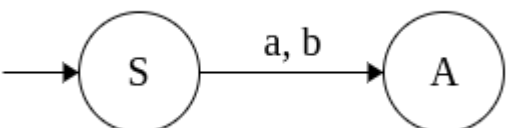
There is a definite advantage with this way of specifying the grammar. This is called as right-linear grammar. It can be easily turned into an NFA. In a right-linear grammar, the right-hand side of a production (or rule) is either a terminal or a terminal followed by a non-terminal.

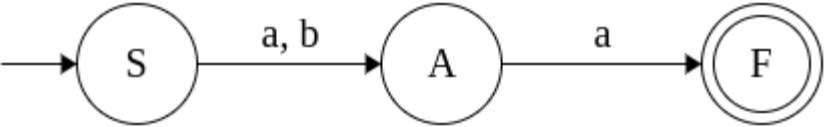
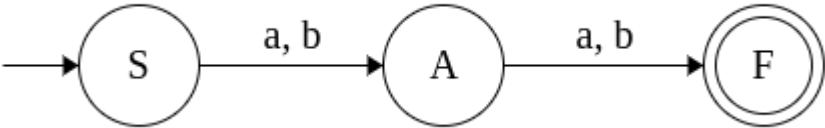
We will now demonstrate the construction of NFA with few examples.

Example 1: The grammar for the regex $(a+b)(a+b)$

$S \rightarrow aA$
 $S \rightarrow bA$
 $A \rightarrow a$
 $A \rightarrow b$

Lets convert the grammar to an equivalent NFA step-by-step.

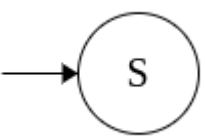
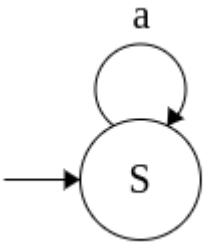
Regular grammar	Equivalent NFA
$S \rightarrow aA$ $S \rightarrow bA$ $A \rightarrow a$ $A \rightarrow b$	 <="" td="">
$S \rightarrow aA$ $S \rightarrow bA$ $A \rightarrow a$ $A \rightarrow b$	 <="" td="">
$S \rightarrow aA$ $S \rightarrow bA$ $A \rightarrow a$ $A \rightarrow b$	 <="" td="">

$S \rightarrow aA$ $S \rightarrow bA$ $A \rightarrow a$ $A \rightarrow b$	
$S \rightarrow aA$ $S \rightarrow bA$ $A \rightarrow a$ $A \rightarrow b$	

Example 2: Grammar for $(a+b)^*abb$

$S \rightarrow aS$
 $S \rightarrow bS$
 $S \rightarrow aB$
 $B \rightarrow bC$
 $C \rightarrow b$

This grammar is converted to equivalent NFA as follows:

Regular grammar	Equivalent NFA
$S \rightarrow aS$ $S \rightarrow bS$ $S \rightarrow aB$ $B \rightarrow bC$ $C \rightarrow b$	
$S \rightarrow aS$ $S \rightarrow bS$ $S \rightarrow aB$ $B \rightarrow bC$ $C \rightarrow b$	

$S \rightarrow aS$ $S \rightarrow bS$ $S \rightarrow aB$ $B \rightarrow bC$ $C \rightarrow b$	
$S \rightarrow aS$ $S \rightarrow bS$ $S \rightarrow aB$ $B \rightarrow bC$ $C \rightarrow b$	
$S \rightarrow aS$ $S \rightarrow bS$ $S \rightarrow aB$ $B \rightarrow bC$ $C \rightarrow b$	
$S \rightarrow aS$ $S \rightarrow bS$ $S \rightarrow aB$ $B \rightarrow bC$ $C \rightarrow b$	

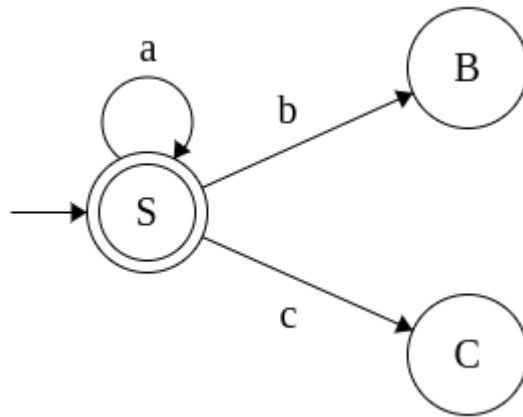
Example 3: Grammar for $a^*b^*c^*$

$S \rightarrow aS$
 $S \rightarrow bB$
 $S \rightarrow cC$
 $S \rightarrow \epsilon$
 $B \rightarrow bB$
 $B \rightarrow cC$
 $B \rightarrow \epsilon$
 $C \rightarrow cC$
 $C \rightarrow \epsilon$

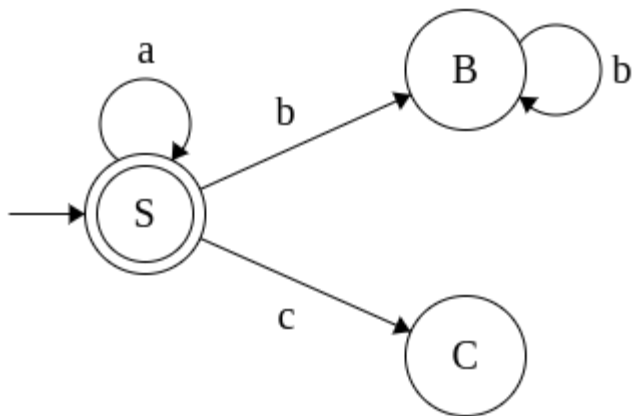
This grammar is converted to equivalent NFA as follows:

Regular grammar	Equivalent NFA
$S \rightarrow aS$ $S \rightarrow bB$ $S \rightarrow cC$ $S \rightarrow \epsilon$ $B \rightarrow bB$ $B \rightarrow cC$ $B \rightarrow \epsilon$ $C \rightarrow cC$ $C \rightarrow \epsilon$	
$S \rightarrow aS$ $S \rightarrow bB$ $S \rightarrow cC$ $S \rightarrow \epsilon$ $B \rightarrow bB$ $B \rightarrow cC$ $B \rightarrow \epsilon$ $C \rightarrow cC$ $C \rightarrow \epsilon$	
$S \rightarrow aS$ $S \rightarrow bB$ $S \rightarrow cC$ $S \rightarrow \epsilon$ $B \rightarrow bB$ $B \rightarrow cC$ $B \rightarrow \epsilon$ $C \rightarrow cC$ $C \rightarrow \epsilon$	
$S \rightarrow aS$ $S \rightarrow bB$ $S \rightarrow cC$ $S \rightarrow \epsilon$ $B \rightarrow bB$ $B \rightarrow cC$ $B \rightarrow \epsilon$ $C \rightarrow cC$ $C \rightarrow \epsilon$	

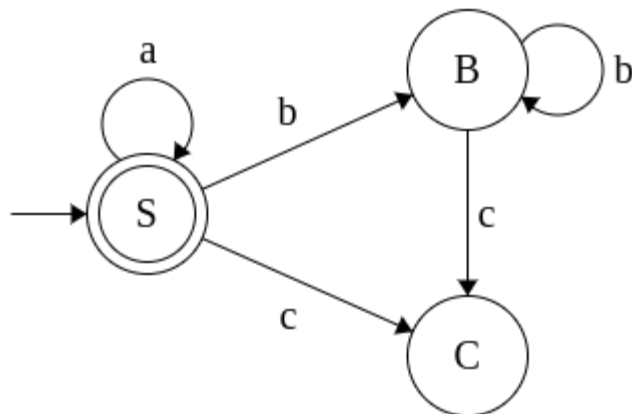
$S \rightarrow aS$
 $S \rightarrow bB$
 $S \rightarrow cC$
 $S \rightarrow \epsilon$
 $B \rightarrow bB$
 $B \rightarrow cC$
 $B \rightarrow \epsilon$
 $C \rightarrow cC$
 $C \rightarrow \epsilon$



$S \rightarrow aS$
 $S \rightarrow bB$
 $S \rightarrow cC$
 $S \rightarrow \epsilon$
 $\mathbf{B} \rightarrow \mathbf{bB}$
 $B \rightarrow cC$
 $B \rightarrow \epsilon$
 $C \rightarrow cC$
 $C \rightarrow \epsilon$



$S \rightarrow aS$
 $S \rightarrow bB$
 $S \rightarrow cC$
 $S \rightarrow \epsilon$
 $B \rightarrow bB$
 $\mathbf{B} \rightarrow \mathbf{cC}$
 $B \rightarrow \epsilon$
 $C \rightarrow cC$
 $C \rightarrow \epsilon$



$S \rightarrow aS$ $S \rightarrow bB$ $S \rightarrow cC$ $S \rightarrow \epsilon$ $B \rightarrow bB$ $B \rightarrow cC$ $\mathbf{B} \rightarrow \epsilon$ $C \rightarrow cC$ $C \rightarrow \epsilon$	<pre> graph LR S((S)) -- a --> S S -- b --> B((B)) S -- c --> C((C)) B -- b --> B B -- c --> C style S stroke-width:4px style B stroke-width:4px </pre>
$S \rightarrow aS$ $S \rightarrow bB$ $S \rightarrow cC$ $S \rightarrow \epsilon$ $B \rightarrow bB$ $B \rightarrow cC$ $B \rightarrow \epsilon$ $\mathbf{C} \rightarrow cC$ $C \rightarrow \epsilon$	<pre> graph LR S((S)) -- a --> S S -- b --> B((B)) S -- c --> C((C)) B -- b --> B B -- c --> C C -- c --> C style S stroke-width:4px style B stroke-width:4px </pre>
$S \rightarrow aS$ $S \rightarrow bB$ $S \rightarrow cC$ $S \rightarrow \epsilon$ $B \rightarrow bB$ $B \rightarrow cC$ $B \rightarrow \epsilon$ $C \rightarrow cC$ $\mathbf{C} \rightarrow \epsilon$	<pre> graph LR S((S)) -- a --> S S -- b --> B((B)) S -- c --> C((C)) B -- b --> B B -- c --> C C -- c --> C style S stroke-width:4px style B stroke-width:4px style C stroke-width:4px </pre>

Properties of Regular Languages

So far we have seen different ways of specifying regular language: DFA, NFA, ϵ -NFA, regular expressions and regular grammar. We noted that all these different expressions are equal in

power by showing the equivalences. Regular expressions and grammars are considered as generators of regular language while the machines (DFA, NFA, ϵ -NFA) are considered as acceptors of the language.

Now we will look at the properties of regular language. The properties can be broadly classified as two parts: (A) Closure properties and (B) Decision properties

(A) Closure Properties

1. Complementation

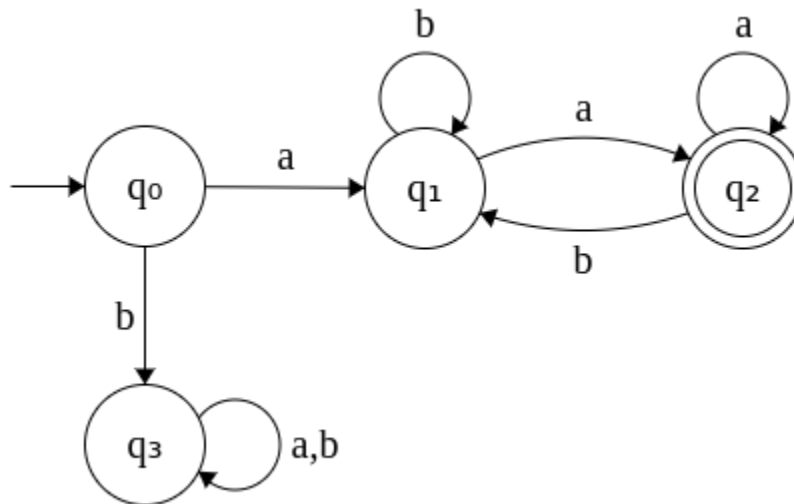
If a language L is regular its complement L' is regular.

Let $DFA(L)$ denote the DFA for the language L . Modify the DFA as follows to obtain $DFA(L')$.

1. Change the final states to non-final states.
2. Change the non-final states to final states.

Since there exists a $DFA(L')$ now, L' is regular.

This can be shown by an example using a DFA. Let L denote the language containing strings that begins and ends with a . $\Sigma = \{a, b\}$. The DFA for L is given below.

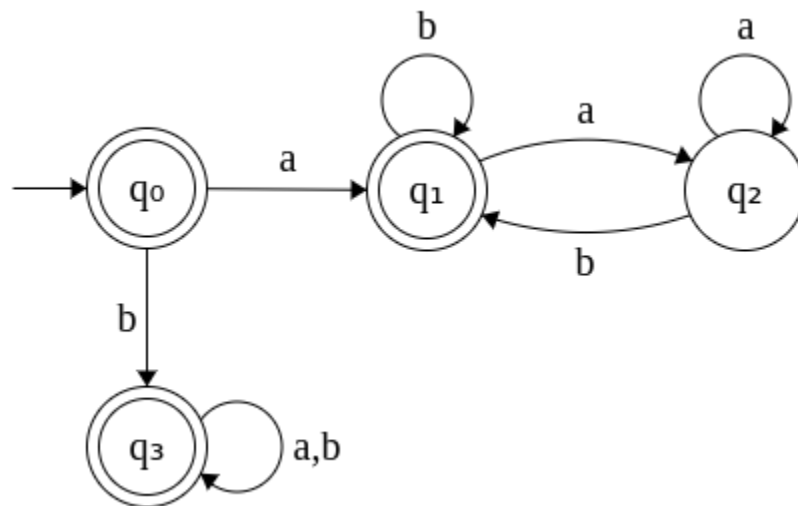


Note: q_3 denotes the dead state.
Once you enter q_3 , you remain in it forever.

L' denotes the language that does not contain strings that begin and end with a . This implies L' contains strings that

- begins with a and ends with b
- begins with b and ends with a
- begins with b and ends with b

The DFA for L' is obtained by flipping the final states of $DFA(L)$ to non-final states and vice-versa. The DFA for L' is given below.



- q_0 ensures ϵ is accepted
- q_1 ensures all strings that begin with a and end with b are accepted.
- q_3 ensures all strings that begin with b (ending with either a or b) are accepted.

Important Note: While specifying the DFA for L , we have also included the dead state q_3 . It is important to include the dead state(s) if we are going to derive the complement DFA since, the dead state(s) too would become final in the complementation. If we didn't add the dead state(s) originally, the complement will not accept all strings supposed to be accepted.

In the above example, if we didn't include q_3 originally, the complement will not accept strings starting with b. It will only accept strings that begin with a and end with b which is only a subset of the complement.

CONCLUSION: REGULAR LANGUAGES ARE CLOSED UNDER COMPLEMENTATION.

2. Union

If L_1 and L_2 are regular, then $L_1 \cup L_2$ is regular.

This is easier proved using regular expressions. If L_1 is regular, there exists a regular expression R_1 to describe it. Similarly, if L_2 is regular, there exists a regular expression R_2 to describe it. $R_1 + R_2$ denotes the regular expression that describe $L_1 \cup L_2$. Therefore, $L_1 \cup L_2$ is regular.

This again can be shown using an example. If L_1 is a language that contains strings that begin with a and L_2 is a language that contain strings that end with a, then $L_1 \cup L_2$ denotes the language the contain strings that either begin with a or end with a.

- $a(a+b)^*$ is the regular expression that denotes L_1 .

- $(a+b)^*a$ is the regular expression that denotes L_2 .

- $L_1 \cup L_2$ is denoted by the regular expression $a(a+b)^* + (a+b)^*a$. Therefore, $L_1 \cup L_2$ is regular.

In terms of DFA, we can say that a $DFA(L_1 \cup L_2)$ accepts those strings that are accepted by either $DFA(L_1)$ or $DFA(L_2)$ or both.

- $DFA(L_1 \cup L_2)$ can be constructed by adding a new start state and new final state.
- The new start state connects to the two start states of $DFA(L_1)$ and $DFA(L_2)$ by ϵ transitions.
- Similarly, two ϵ transitions are added from the final states of $DFA(L_1)$ and $DFA(L_2)$ to the new final state.
- Convert this resulting NFA to its equivalent DFA.

As an exercise you can try this approach of DFA construction for union for the given example.

CONCLUSION: REGULAR LANGUAGES ARE CLOSED UNDER UNION.

3. Intersection

If L_1 and L_2 are regular, then $L_1 \cap L_2$ is regular.

Since a language denotes a set of (possibly infinite) strings and we have shown above that regular languages are closed under union and complementation, by De Morgan's law can be applied to show that regular languages are closed under intersection too.

L_1 and L_2 are regular $\Rightarrow L_1'$ and L_2' are regular (by Complementation property)

$L_1' \cup L_2'$ is regular (by Union property)

$L_1 \cap L_2$ is regular (by De Morgan's law)

In terms of DFA, we can say that a $DFA(L_1 \cap L_2)$ accepts those strings that are accepted by both $DFA(L_1)$ and $DFA(L_2)$.

CONCLUSION: REGULAR LANGUAGES ARE CLOSED UNDER INTERSECTION.

4. Concatenation

If L_1 and L_2 are regular, then $L_1 \cdot L_2$ is regular.

This can be easily proved by regular expressions. If R_1 is a regular expression denoting L_1 and R_2 is a regular expression denoting L_2 , then we $R_1 \cdot R_2$ denotes the regular expression denoting $L_1 \cdot L_2$. Therefore, $L_1 \cdot L_2$ is regular.

In terms of DFA, we can say that a $DFA(L_1 \cdot L_2)$ can be constructed by adding an ϵ -transition from the final state of $DFA(L_1)$ - which now ceases to be the final state - to the start state of $DFA(L_2)$. You can try showing this using an example.

CONCLUSION: REGULAR LANGUAGES ARE CLOSED UNDER CONCATENATION.

5. Kleene star

If L is regular, then L^* is regular.

This can be easily proved by regular expression. If L is regular, then there exists a regular expression R . We know that if R is a regular expression, R^* is a regular expression too. R^* denotes the language L^* . Therefore L^* is regular.

In terms of DFA, in the $DFA(L)$ we add two ϵ transitions, one from start state to final state and another from final state to start state. This denotes $DFA(L^*)$. You can try showing this for an example.

CONCLUSION: REGULAR LANGUAGES ARE CLOSED UNDER KLEENE STAR.

6. Difference

If L_1 and L_2 are regular, then $L_1 - L_2$ is regular.

We know that $L_1 - L_2 = L_1 \cap L_2'$

L_1 and L_2 are regular $\Rightarrow L_1$ and L_2' are regular (by Complementation property)

$L_1 \cap L_2'$ is regular (by Intersection property)

$L_1 - L_2$ is regular (by De Morgan's law)

In terms of DFA, we can say that a $DFA(L_1 - L_2)$ accepts those strings that are accepted by both $DFA(L_1)$ and not accepted by $DFA(L_2)$. You can try showing this for an example.

CONCLUSION: REGULAR LANGUAGES ARE CLOSED UNDER DIFFERENCE.

7. Reverse

If L is regular, then L^R is regular.

Let $DFA(L)$ denote the DFA of L . Make the following modifications to construct $DFA(L^R)$.

1. Change the start state of $DFA(L)$ to the final state.
2. Change the final state of $DFA(L)$ to the start state.

In case there are more than one final state in DFA(L), first add a new final state and add ϵ -transitions from the final states (which now cease to be final states any more) and perform this step.

3. Reverse the direction of the arrows.

You can try showing this using an example.

CONCLUSION: REGULAR LANGUAGES ARE CLOSED UNDER REVERSAL.

Pumping lemma and its applications

Not all languages are regular. For example, the language $L = \{a^n b^n : n \geq 0\}$ is not regular. Similarly, the language $\{a^p : p \text{ is a prime number}\}$ is not regular. A pertinent question therefore is how do we know if a language is not regular.

Question: Can we conclude that a language is not regular if no one could come up with a DFA, NFA, ϵ -NFA, regular expression or regular grammar so far?

- No. Since, someone may very well come up with any of these in future.

We need a property that just holds for regular languages and so we can prove that any language without that property is not regular. Let's recall some of the properties.

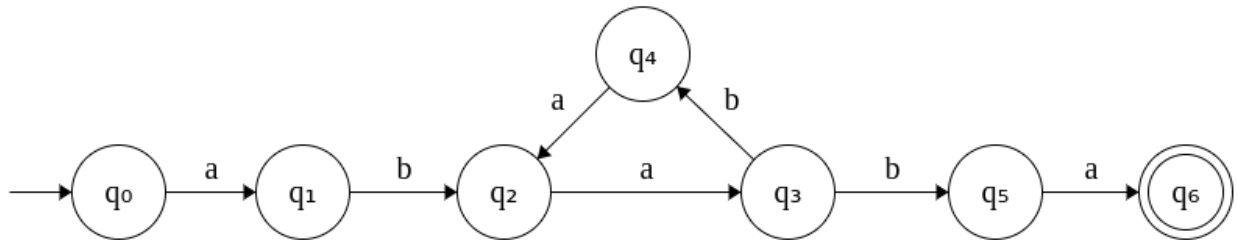
- We have seen that a regular language can be expressed by a finite state automaton. Be it deterministic or non-deterministic, the automaton consists of a finite set of states.
- Since the states are finite, if the automaton has no loop, the language would be finite.
 - Any finite language is indeed a regular language since we can express the language using the regular expression: $S_1 + S_2 + \dots + S_N$, where N is the total number of strings accepted by the automaton.
- However, if the automaton has a loop, it is capable of accepting infinite number of strings.
 - Because, we can loop around any number of times and keep producing more and more strings.
 - This property is called the pumping property (elaborated below).

The pumping property of regular languages

Any finite automaton with a loop can be divided into parts three.

- Part 1: The transitions it takes before the loop.
- Part 2: The transitions it takes during the loop.
- Part 3: The transitions it takes after the loop.

For example consider the following DFA. It accepts all strings that start with aba followed by any number of baa's and finally ending with ba.

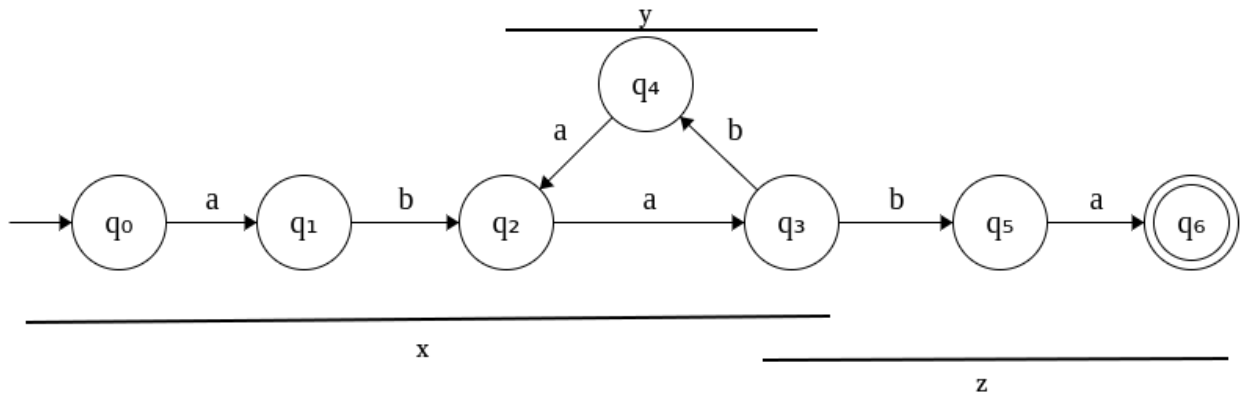


1. What strings are accepted by this DFA?

abab**a**aba, abab**a**abab**a**ba, abab**a**abababab**a**ba, so on and so forth. Thus the strings accepted by the above DFA can be divided into three parts: **aba**, **(baa)ⁱ** and **ba**. Here, $i > 0$.

Investigating this further, we can say that any string w accepted by this DFA can be written as $w = x y^i z$

where y represents the part that can be pumped again and again to generate more and more valid strings. This is shown below for the given example.



Before we generalize further, let's investigate this example a little more.

2. What if the loop was at the beginning? Say a self-loop at q_0 instead of at q_2 .

Then $x = \epsilon$ or $|x| = 0$. In such a special case, $w = yz$.

3. What if the loop was at the end. Say a self loop at q_6 instead of at q_2 .

Then $z = \epsilon$ or $|z| = 0$. In such a special case, $w = xy$.

4. Can y be equal to ϵ ever?

No. It is impossible. If $y = \epsilon$, it implies there is no loop which implies the language is finite.

We have already seen that a finite language is always regular. So, we are now concerned only with infinite regular language. Hence, y can never be ϵ . Or $|y| > 0$.

5. What is the shortest string that is accepted by the DFA?
ababa. Obviously, a string obtained without going through the loop.
There is a catch however. See the next question.
6. What is the shortest string accepted if there are more final states? Say q_2 is final.
ab of length 2.
7. What is the longest string accepted by the DFA without going through the loop even once?
ababa ($= xz$). So, any string of length > 5 accepted by DFA must go through the loop at least once.
8. What is the longest string accepted by the DFA by going through the loop exactly once?
ababaaba ($= xyz$) of length 8. We call this pumping length.

More precisely, pumping length is an integer p denoting the length of the string w such that w is obtained by going through the loop exactly once. In other words, $|w| = |xyz| = p$.

9. Of what use is this pumping length p ?
We can be sure that $|xy| \leq p$. This can be used to prove a language non-regular.

Now, let's define a regular language based on the pumping property.

Pumping Lemma: If L is a regular language, then there exists a constant p such that every string $w \in L$, of length p or more can be written as $w = xyz$, where

1. $|y| > 0$
2. $|xy| \leq p$
3. $xy^iz \in L$ for all i

Proving languages non-regular

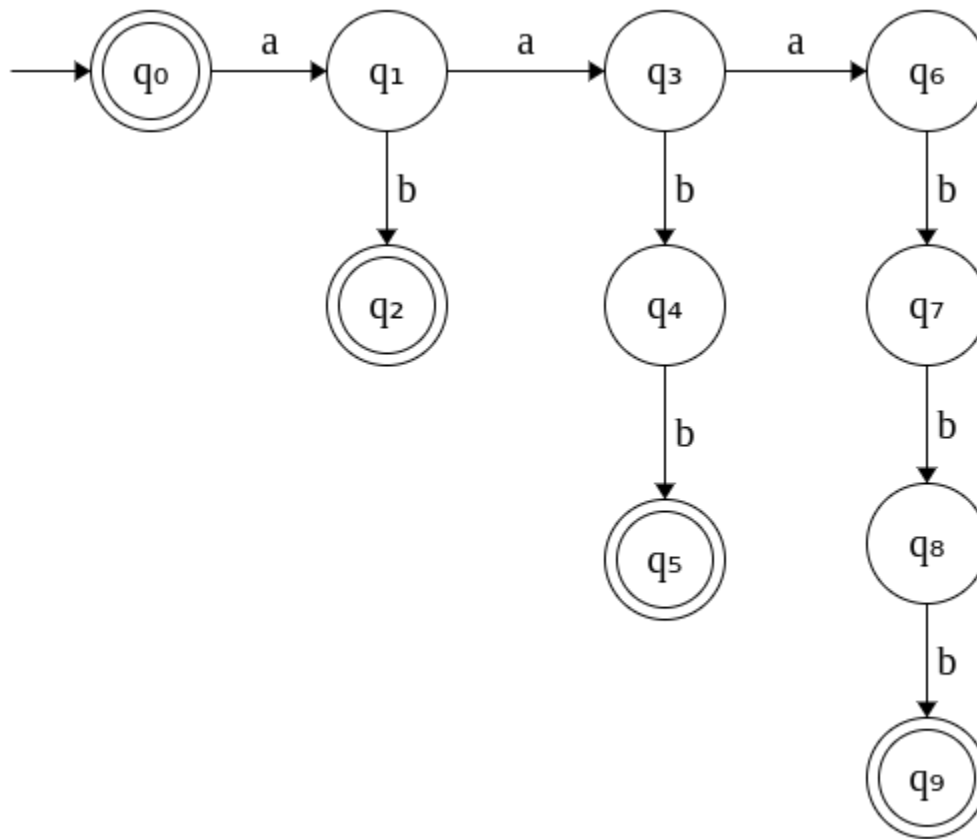
1. The language $L = \{ a^n b^n : n \neq 0 \}$ is not regular.

Before proving L is not regular using pumping property, let's see why we can't come up with a DFA or regular expression for L .

$$L = \{ \epsilon, ab, aabb, aaabbb, \dots \}$$

It may be tempting to use the regular expression a^*b^* to describe L . No doubt, a^*b^* generates these strings. However, it is not appropriate since it generates other strings not in L such as a , b , aa , ab , aaa , aab , abb , ...

Let's try to come up with a DFA. Since it has to accept ϵ , start state has to be final. The following DFA can accept $a^n b^n$ for $n \leq 3$. i.e. $\{\epsilon, a, b, ab, aabb, aaabbb\}$



The basic problem is DFA does not have any memory. A transition just depends on the current state. So it cannot keep count of how many a's it has seen. So, it has no way to match the number of a's and b's. So, only way to accept all the strings of L is to keep adding newer and newer states which makes automaton to infinite states since n is unbounded.

Now, let's prove that L does not have the pumping property.

Lets assume L is regular. Let p be the pumping length.

Consider a string $w = aa....abb....b$ such that $|w| = p$.

$$\Rightarrow w = a^{p/2} b^{p/2}$$

We know that w can be broken into three terms xyz such that $y \neq \epsilon$ and $xy^i z \in L$.

There are three cases to consider.

- Case 1: y is made up of only a's

Then xy^2z has more a's than b's and does not belong to L.

- Case 2: y is made up of only b's

Then xy^2z has more b's than a's and does not belong to L.

- Case 3: y is made up of a's and b's

Then xy^2z has a's and b's out of order and does not belong to L.

Since none of the 3 cases hold, the pumping property does not hold for L. And therefore L is not regular.

2. The language $L = \{ uu^R : u \in \{a,b\}^* \}$ is not regular.

Lets assume L is regular. Let p be the pumping length.

Consider a string $w = a^p b b a^p$.

$$|w| = 2p + 2 \geq p$$

Since, $xy \leq p$, xy will consist of only a's.

\Rightarrow y is made of only a's

$\Rightarrow y^2$ is made of more number of a's than y since $|y| > 0$

(Let's say y^2 has m a's more than y where $m > 1$)

$\Rightarrow xy^2z = a^{p+m} b b a^p$ where $m \geq 1$

$\Rightarrow xy^2z = a^{p+m} b b a^p$ cannot belong to L.

Therefore, pumping property does not hold for L. Hence, L is not regular.

3. The language $L = \{ a^n : n \text{ is prime} \}$ is not regular.

Lets assume L is regular. Let p be the pumping length. Let $q \geq p$ be a prime number (since we cannot assume that pumping length p will be prime).

Consider the string $w = aa \dots a$ such that $|w| = q \geq p$.

We know that w can be broken into three terms xyz such that $y \neq \epsilon$ and $xy^i z \in L$

$\Rightarrow xy^{q+1}z$ must belong to L

$\Rightarrow |xy^{q+1}z|$ must be prime

$$|xy^{q+1}z| = |xyzy^q|$$

$$= |xyz| + |y^q|$$

$$= q + q \cdot |y|$$

$$= q(1 + |y|) \text{ which is a composite number.}$$

Therefore, $xy^{q+1}z$ cannot belong to L . Hence, L is not regular.

Exercises

Show that the following languages are not regular.

4. $L = \{ a^n b^m : n \neq m \}$

5. $L = \{ a^n b^m : n > m \}$

6. $L = \{ w : n_a(w) = n_b(w) \}$

7. $L = \{ ww : w \in \{a,b\}^* \}$

8. $L = \{ a^{n^2} : n > 0 \}$

IMPORTANT NOTE

Never use pumping lemma to prove a language regular. Pumping property is necessary but not sufficient for regularity.